DEPARTEMENT INFORMATIQUE – IUT2 GRENOBLE

Année Universitaire 2003-2004

MEMOIRE DE STAGE

REALISATION D'UN LOGICIEL D'AIDE A LA CREATION D'UN RESEAU BIOCHIMIQUE

Laboratoire TIMC – Equipe TIMB

(Stage du 13 Avril au 20 Juin 2004)

Présenté par

Benjamin Millau

Jury

IUT : Mme Rieu
IUT : M.Gérot
Société : M. Martiel
Société : M. Berro

REMERCIEMENTS

Jean-Louis Martiel
Directeur de l'équipe TIMB

Julien Berro
Etudiant thésard

pour m'avoir accueilli lors de ce stage.

SOMMAIRE

Remerciements……2
Sommaire……3
Introduction……4
Le laboratoire TIMC……4
L'équipe TIMB……4
Le sujet du stage……5
Analyse……6
Cahier des charges……6
Fonctionnalités du logiciel……6
Contrainte……8
Etapes de développement……9
Matériel et outils utilisés……11
Conception informatique……12
Réalisation……24
Etapes de réalisation……25
La classe Network……26
Les autres classes……35
Mise en fonction……36
Etat d'avancement du projet……36
Test effectués et manuel d'utilisateur……36
Problèmes rencontrés……37
Modifications effectués et prise en compte des remarques des utilisateurs……38
Conclusion……39
Glossaire……40
Bibliographie……41
Annexe 1 : le planning de travail……43
Annexe 2 : la méthode setMolecules……44
Annexe 3 : la méthode getRéactionList……46
Annexe 4 : manuel d'utilisateur……50
Annexe 5 : spécification de la base de données par J.Berro……59
Annexe 6 : fenêtre de Jdesigner……80

INTRODUCTION

Le laboratoire TIMC

Le laboratoire TIMC(UMR-CNRS 5525), implanté sur le campus Santé du CHU de Grenoble, représente l'interface de l'IMAG ouverte sur les applications biologiques et médicales des mathématiques et de l'informatique. Les activités de TIMC couvrent un très large spectre de l'ingénierie de l'information en santé.
- la méthodologie de modélisation et de structuration informatiques de la connaissance médicale (OSIRIS), la modélisation mathématique des processus cognitifs et des processus de prolifération, migration et biomécanique cellulaires (DynaCell, TIMB, RFMQ).
- la bio-informatique du génome humain (GENOME), les techniques d'IA et la décision médicale (SIC), le traitement et l'analyse d'images médicales et microscopiques, la robotique chirurgicale (GMCAO) .
- l'acquisition guidée par le modèle d'observables biomédicaux (images et signaux)(AFIRM).
- la surveillance des maladies environnementales émergentes (EPSP).
- l'étude quantitative de l'activité respiratoire et de l'activité électro-encéphalographique au cours de la veille et du sommeil (PRETA).

L'équipe TIMB

Le but de la recherche à TIMB est d'étudier et de développer des modèles mathématiques en étroite connexion avec des problèmes biologiques et médicaux ou des applications. Il faut donc intégrer des aspects fondamentaux de biologie, dont les mécanismes moléculaire et cellulaire, la formation de tissu, la génétique, le raisonnement médical dans le cadre d'un modèle mathématique. L'analyse du modèle est effectuée par analyse théorique et simulation numérique des équations.

Le sujet du stage

Le but du stage est de développer une interface d'aide à la création de réseaux de régulation biochimique, à partir de données biochimiques stockées dans une base de donnée distante. Un réseau de régulation biochimique est un graphe permettant de représenter les interactions chimiques entre différentes molécules dans une partie d'un organisme vivant.

Dans un premier temps, le logiciel sera une aide au remplissage des différents champs nécessaires à la création d'un réseau dans le logiciel JDesigner (logiciel gratuit que l'on peut obtenir sur le site: http://www.cds.caltech.edu/~hsauro/JDesigner.htm). Dans un deuxième temps, il devra permettre de générer directement un fichier SBML (spécifications du format sur le site: http://sbml.org/documents/) de description du réseau de régulation. Un fichier SBML (Systems Biology Markup Language) est un fichier de type « fichier à balise », comme un fichier XML, qui permet de spécifier un réseau de régulation biochimique.

L'interface devra être très simple et très « user-friendly », un des objectifs étant de distribuer ce logiciel à des biologistes n'ayant pas forcement de connaissance très poussée en informatique. D'autre part, il sera nécessaire que l'interface puisse fonctionner sur plusieurs plateformes, c'est à dire sur Linux, Windows, voir Mac éventuellement).

La base de donnée est logée physiquement sur une machine de TIMC et peut être utilisée à l'extérieur de TIMC. Il faudra donc gérer les problèmes de sécurité et d'accès à la base de données.

Une procédure d'installation et de configuration simple devra être mis en place, comme une installation automatique du logiciel, ainsi que des clients PostgreSQL et de tout ce qui est nécessaire.

Afin de correspondre au mieux au logiciel JDesigner, le logiciel que nous allons réaliser possède une interface semblable à celle de ce dernier. Ainsi, l'utilisateur pourra faire le choix entre plusieurs molécules, puis, à partir de ces molécules, pourra sélectionner des réactions et autre lois cinétiques disponibles dans la base de données. Ensuite, l'utilisateur n'aura plus qu'à recopier les informations dans les champs de JDesigner prévu à cet effet.
Enfin, le logiciel permettra de sauvegarder directement les informations sélectionnées et, s'il le désire, éditées, dans un fichier au format SBML.

Pour réaliser ce logiciel, nous avons choisi le langage Java, qui présente l'avantage d'être très portable, et possède une librairie d'interface graphique de bonne qualité et simple d'utilisation : SWING. La base de donnée sera contrôlée grâce à un pilote PostgreSQL. Afin de pouvoir gérer aux mieux les différentes version concurrentes du logiciel, nous avons utilisé un serveur CVS, étant donné que nous sommes deux étudiants stagiaires pour réaliser ce projet.

Analyse

Cahier des charges

Fonctionnalités du logiciel:

Le logiciel doit permettre d'être facilement compris par tous les usagers, l'interface doit donc être simple d'utilisation. Celle-ci doit reprendre l'aspect et tous les champs de celles du logiciel gratuit Jdesigner (voir Annexe 6, page 80). Le but du logiciel sera d'aider tous les usagers à construire un réseau de régulation biochimique à partir des informations contenues dans la base de données PathwayDB.

La base de données PathwayDB, dont on peut trouver la spécification en annexe (Annexe 5, page 59), contient différentes notions utilisées dans le cahier des charges.
Tout d'abord, les organismes, qui sont des organismes vivants, tel l'homme, le rat, etc. Chaque organisme peut appartenir à une « famille » d'organismes, nommée taxon, par exemple les mammifères.
Il existe ensuite, dans chaque organisme, différentes parties qu'on nommera locations, et qui découpent donc l'organisme, par exemple dans l'organisme « Homme », on trouvera la location « neurone ».
Enfin, ces locations peuvent encore être décrites géométriquement par des compartiments, qui sont de taille précise et vont permettre de calculer avec une quantité ou une concentration d'éléments.

Ce sont donc les conteneurs de la base de données. Les éléments contenus dans ceux-ci sont des molécules, différenciées dans la base de données par leur numéro d'identification unique. Chaque molécule dispose d'un nom, et peut avoir plusieurs « alias », ou molécules similaires mais de nom différent, qui se rapportent à la même molécule. Chaque molécule peut être associé à différentes concentration initiale, selon la location et le compartiment dont elle est issue. On a donc de nombreuses possibilités selon tous les choix différents.

Enfin les réactions sont stockées dans la base de données. Chaque réaction dispose d'une expression symbolique, de réactifs et de produits, c'est à dire les différentes molécules qui interviennent dans la réaction. Une réaction peut intervenir entre plusieurs molécules de locations ou compartiments différents.
Chaque réaction dispose de différentes lois cinétiques, qui régissent la manière dont les molécules interagissent entre elles lorsque la réaction se produit. Il y a plusieurs lois cinétiques possibles pour une réaction, il faudra donc en choisir une.

Plusieurs options doivent s'offrir a l'usagers : tout d'abord, afin de pouvoir fabriquer un réseau de régulation biochimique, l'usager doit pouvoir choisir dans quel domaine il souhaite travailler. Il peut sélectionner au début:
- soit par organisme, par exemple le rat.
- soit par taxon, c'est à dire une famille d'organisme, comme par exemple les mammifères.
- soit il peut avoir comme choix de construire un réseau sans spécifier de domaine particulier pour travailler.

Et l'utilisateur fait son choix parmi les différents organismes ou autres taxons proposés, ou passe directement au choix des molécules, s'il ne veut travailler dans aucun domaine précis.

Ensuite, l'utilisateur doit pouvoir choisir une première location dans laquelle il souhaite travailler. La liste des locations doit uniquement afficher les locations pour l'organisme ou le taxon sélectionné selon le premier choix. Cette étape ne s'effectuera pas si l'utilisateur a choisi de ne pas spécifier de domaine de travail particulier.

Puis, l'utilisateur, sélectionne un compartiment dans lequel il souhaite travailler. Lors du choix du compartiment, doit apparaître le nom du compartiment, la taille de ce compartiment et les diverses informations préalablement rentrées dans la base de données.
Comme pour l'étape précédente, celle ci ne s'effectuera pas si l'utilisateur n'a pas spécifié de domaine de travail.

Après cela, l'utilisateur doit pouvoir sélectionner les molécules qu'il souhaite intégrer dans le réseau à partir de la liste complète de toutes les molécules entrées dans la base de donnée PathwayDB quelque soient les choix effectués précédemment. Il doit pouvoir sélectionner un nombre indéterminé de molécules. Les molécules possédant un ou des alias doivent être précisées et ne doivent pas être sélectionnées plusieurs fois.

Pour continuer, la liste des réactions possibles entre les molécules choisis disponible dans la base de données, doivent s'afficher. L'utilisateur doit pouvoir sélectionner seulement celles qui l'intéresse. Il doit également pouvoir sélectionner toutes les réactions s'il le désire. Les réactions seront affichées à partir de leurs expressions symboliques.

Une fois l'utilisateur ayant validé tous ses choix, la fenêtre principale du logiciel doit être affichée. Celle-ci doit contenir quasiment toutes les informations du réseau de régulation biochimique. C'est cette fenêtre qui doit être similaire a celles de JDesigner, et elle doit contenir les informations suivante:
- la liste de toutes les molécules sélectionnées, avec leur alias et le compartiment dans lequel elle se situe.
- la liste des différents paramètres de la molécule. Une molécule doit pouvoir être mis en surbrillance afin de pouvoir faire le choix entre ses différents paramètres. Ces paramètres sont la concentration ou la quantité (les 2 doivent pouvoir être afficher, mais un seul sera pris un compte pour la suite) et le statut. La quantité étant contenue dans la base de donnée PathwayDB, et la concentration étant la quantité divisée par la taille du compartiment dans laquelle se trouve la molécule. La référence correspondant au paramètre de la molécule choisie doit également être affichée.
- la liste de toutes les réactions sélectionnées précédemment. Une de ces réactions doit pouvoir être misent en surbrillance afin de pouvoir faire le choix parmi ses lois cinétiques. La véritable expression de la réaction doit également être affiché, montrant ainsi les stœchiométries des molécules mises en jeu.
- la liste des lois cinétiques disponibles pour la réaction sélectionnée doit être affichée et également mis en surbrillance. Comme pour les réactions, ce doit être les expressions symboliques qui doivent permettre de sélectionner la loi cinétique.
- les jeux de paramètres disponibles pour la loi cinétique choisie doivent également apparaître et être sélectionnable. La référence correspondant au jeu de paramètres choisi doit être affichée.
Deux choix ensuite doivent se proposer à l'utilisateur: il peut soit choisir de continuer, soit de s'arrêter.
Si, au départ, l'utilisateur avait choisi de ne pas spécifier de domaine, alors, il ne peut pas continuer.

S'il choisit de continuer, alors il va de nouveau pouvoir choisir une location, différente ou non, et un compartiment forcément différent, puis choisir les molécules qu'il veut avoir dans ce compartiment. Pour le choix des réactions, toutes les réactions possibles doivent être sélectionnables, même si les molécules ne se trouvent pas dans le même compartiment (il peut y

avoir des réactions inter-compartiments, et donc inter-locations aussi) . Ensuite, la fenêtre principale doit se réafficher, contenant toutes les informations déjà sélectionnées plus les nouvelles. Ainsi, si une molécule a été sélectionnée deux fois dans des compartiments différents, elle doit apparaître en double, avec le nom du compartiment dans lequel elle se situe, et donc des paramètres différents. En ce qui concerne les réactions, toutes les réactions qui ont été sélectionnées doivent être affichées. L'utilisateur doit pouvoir continuer jusqu'à ce qu'il n'y ait plus de compartiments qui n'a pas déjà été sélectionné.
Tous les choix effectués en ce qui concerne les choix des paramètres pour les molécules et les choix de lois cinétiques pour les réactions doivent être sauvegardés.

Une fois tous les choix effectués, l'utilisateur peut terminer ses choix en cliquant sur le bouton approprié. Ici, une nouvelle fenêtre ressemblant à la fenêtre principale doit s'afficher. Cette fenêtre doit afficher uniquement les choix de paramètres, de molécules et de réactions, sélectionnés dans la fenêtre précédente, à la différence près qu'il doit pouvoir être libre d'éditer les paramètres. Il faut également qu'il puisse revenir à la fenêtre précédente, afin qu'il puisse revenir sur ses choix. Une fois, l'utilisateur satisfait de ces choix, il va pouvoir, dans un premier temps, copier tous ses paramètres dans les champs prévus à cet effet dans le logiciel JDesigner , puis, lorsque la libraire SBML pour Java sera disponible (normalement, fin Avril), sauvegarder un fichier au format SBML directement utilisable avec les logiciel utilisant ce standard.

Contrainte:

Le logiciel doit être portable, c'est à dire qu'il doit pouvoir fonctionner sous Linux et sous Windows, et éventuellement, sous Macintosh.
La base de données PathwayDB est logée physiquement sur une machine de TIMC. Il faudra donc gérer les problèmes de sécurité d'accès à la base.
Il faudra aussi que le logiciel possède une procédure d'installation simple et rapide, ainsi que tout ce qui pourrait être nécessaire au bon fonctionnement du logiciel, et ce pour tous les supports avec lesquels le logiciel sera compatible.
Bien évidemment, le logiciel devra être au maximum débogué à l'aide de jeux de test. Une documentation détaillée du fonctionnement du logiciel doit également être faite, et le code doit être bien commenté. Enfin, par souci d'internationalisation, le logiciel doit être réalisé en anglais.

Etapes du développement

Etape 0: Installation et configuration des machines de développement.

Dans un premier temps, il faudra installer (ou vérifier qu'ils existent bien) et configurer sur chaque machine de développement:
- un client PostgreSQL
- un logiciel CVS et une interface du type tkCVS
- un environnement de développement pour le langage qu'on aura choisi
- les bibliothèques PostgreSQL pour le langage suscité

Bien évidement, il faudra tester que tous ces logiciels fonctionnent bien, y compris avec des machines distantes (pour CVS et PostgreSQL)

Etape 1: Accès à une base de données distante

La toute première étape sera de faire une première petite interface qui permette de se connecter à une base de donnée sur une machine distante, de gérer les accès sécurisés, les identifications, les droits de consultation ... L'objectif de cette étape étant de pouvoir consulter des données dans la table 'Molecule'.

Dans chacune des étapes, il faudra penser à stocker une trace de tous les choix que l'on a fait (dans un fichier de log par exemple).

Etape 2 : Aide au remplissage de la fenêtre des propriétés des molécules

L'idée est qu'à la fin de cette étape on puisse rentrer dans la fenêtre de propriétés de JDesigner tous les champs relatifs à une molécule de façon très simple. Il faudrait donc, après avoir choisi une molécule, afficher les données dans une fenêtre ressemblant à la fenêtre des propriétés de JDesigner.

Etape 3: Aide au remplissage de la fenêtre des propriétés des réactions

C'est le même esprit que l'étape précédente mais avec les réactions. Il faudrait faire un tableau contenant les molécules de la base et on sélectionnerait les molécules qui nous intéressent. A partir de là, il faudrait proposer plusieurs réactions possibles avec ces molécules, et choisir celle qui convient. Ensuite, il faudrait récupérer les lois cinétiques, les paramètres,...

Etape 4: Ajout de la sélection d'organisme, de locations et compartiments.

Il faudrait pouvoir rajouter un domaine de travail, et donc gérer les différentes concentrations ou quantités des molécules en fonction de la location et du compartiment sélectionné.

Etape 5: Stockage des informations entrées dans les différentes fenêtres de propriétés.

L'idée est de stocker les différentes valeurs employées pour la construction du réseau. L'objectif étant de pouvoir générer un fichier au format SBML à partir de ces données. On utilisera pour ça la bibliothèque libSBML. Il faudrait aussi penser à un affichage récapitulatif des données sélectionnées.

Etape 6: Création d'un réseau.
A la fin de cette étape, on veut avoir un logiciel qui permette de créer un réseau de réaction. Ceci revient en fait à faire une boucle sur le logiciel obtenu à l'étape précédente, et à stocker toute l'information au format SBML. Il faudra prévoir aussi un affichage des données sélectionnées et la possibilité de les modifier.

Etape 7:Ajout de la fonctionnalité de lecture d'un fichier SBML.

Il faudra pouvoir lire un fichier SBML déjà existant et afficher les données qu'il contient dans un format compréhensible comme un ficher texte.

Materiel et outils utilisé.

Afin de réaliser ce logiciel, nous disposons de deux ordinateurs, celui m'ayant été attribué possède comme système d'exploitation Linux Mandrake 9.0 avec comme gestionnaire de fenêtre KDE 3.0. Dans un premier temps, il était nécessaire de configurer nos comptes sur les machines que l'ont allait utiliser.
La base de données PathwayDB pour laquelle le logiciel doit être réalisé est une base de données relationnelle en PostgreSQL. Il était donc important que nous nous familiarisions avec cette base de donnée. Il a donc fallut que nous installions un client PostgreSQL. Nous avons choisi d'utiliser « psql », très bon client PostgreSQL que nous connaissions déjà, et de plus est disponible dans les package de Linux Mandrake 9.0.
Ensuite, après avoir choisi le langage dans lequel le logiciel allait être développé, le Java, il fallait choisir un logiciel adéquat, nous avons donc opté pour le logiciel de Borland, Jbuilder , puissant logiciel de programmation Java, possédant une excellente interface de création de fenêtre et une très grande documentation intégré, et de plus nous connaissions déjà très bien ce logiciel. Nous avons pu trouver ce logiciel sur le site officiel de Borland.
Puis, afin de faire correspondre le logiciel en Java est la base de données PathwayDB en PostgreSQL, il nous fallait une librairie de gestion de base de donnée pour Java. Nous avons donc eu besoin de chercher sur Internet différentes informations à ce propos, et il apparaissait que JBDC pouvait tout à fait correspondre à nos besoins. JDBC (Java DataBase Connectivity) est une bibliothèque Java permettant un dialogue entre un programme Java est une base de données (spécification et documentation sur le site : http://java.sun.com/products/jdbc/).
Nous avons donc choisi d'utiliser cette librairie pour réaliser notre logiciel. Nous n'avons pas eu de mal à trouver cette librairie sur divers site Internet (voir bibliographie).

Un temps d'adaptation fut quand même nécessaire pour l'installation de logiciel et autres packages, n'ayant jamais utilisé Linux ailleurs qu'à l'IUT auparavant, et donc n'ayant pas vraiment l'expérience dans la configuration et de l'installation de logiciel sous Linux. Aussi, certain logiciel demandait certains packages afin de fonctionner, il a fallut donc chercher ces packages sur divers site Internet, ou encore dans les packages fourni avec Linux Mandrake 9.0.
Afin de configurer les deux machines, il a bien fallut environ trois jours pour bien configurer et installer tout le nécessaire sur les deux machines, puisque certain problème de connexion au réseau se sont présentés les deux premiers jours du stage.

Conception informatique

Diagramme de cas d'utilisation

Nous avons tout d'abord identifié les différents cas d'utilisation pouvant se présenter à l'utilisateur. Il y a donc :

Connect – La phase de connexion à la base de données, qui permet donc à l'utilisateur de s'identifier.

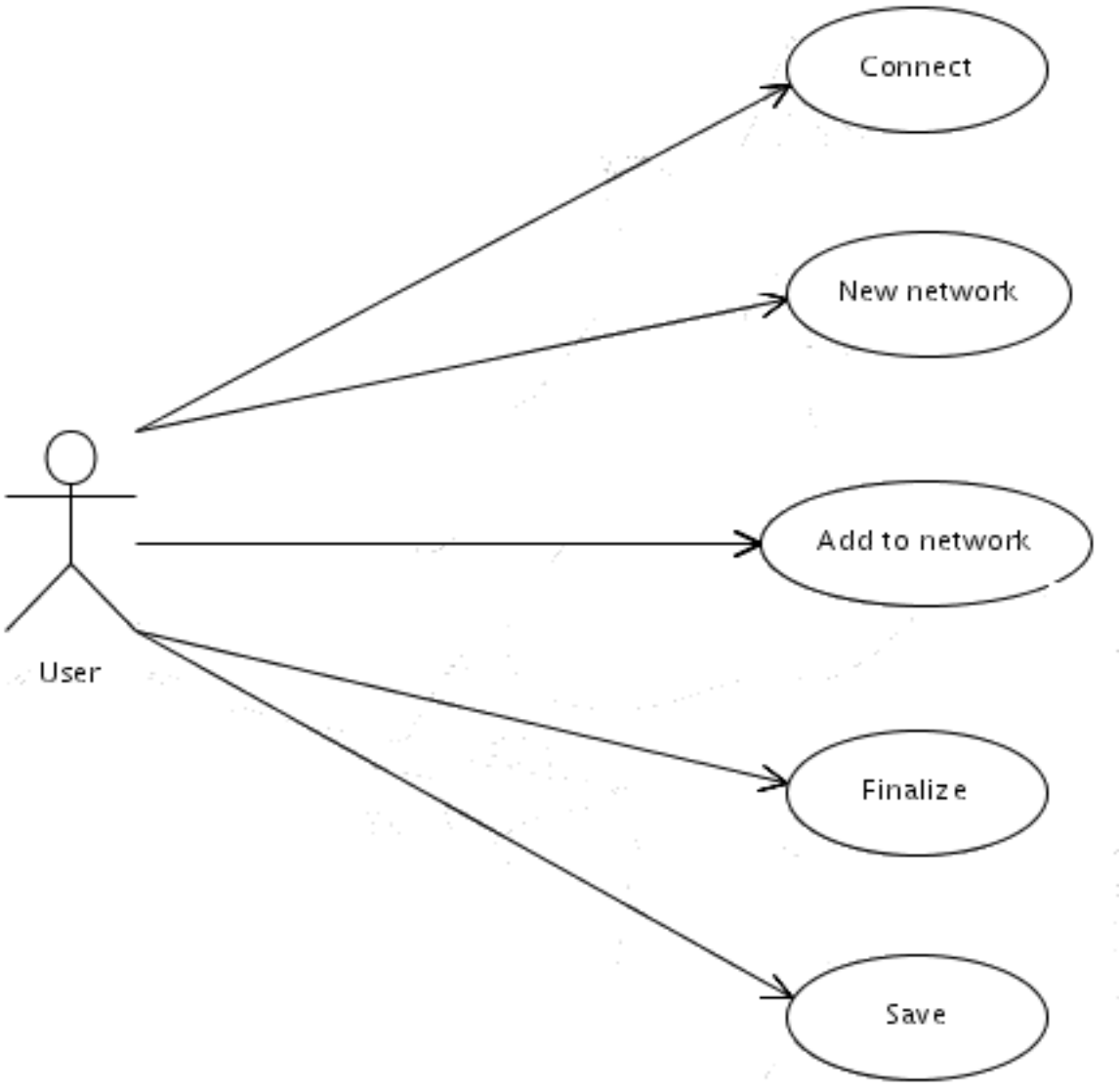
New Network – La phase de création d'un réseau biochimique. L'utilisateur choisit donc la liste des molécules, réactions qu'il veut.

Add to network – La phase de complétion du diagramme : rajouter des molécules / réactions au réseau.

Finalize – La phase de finalisation du programme, qui consiste juste en la modification possible des valeurs manuellement.

Save – La phase finale du programme, sauvegarde du réseau au format de fichier SBML.

On obtient donc le diagramme de cas d'utilisation suivant :

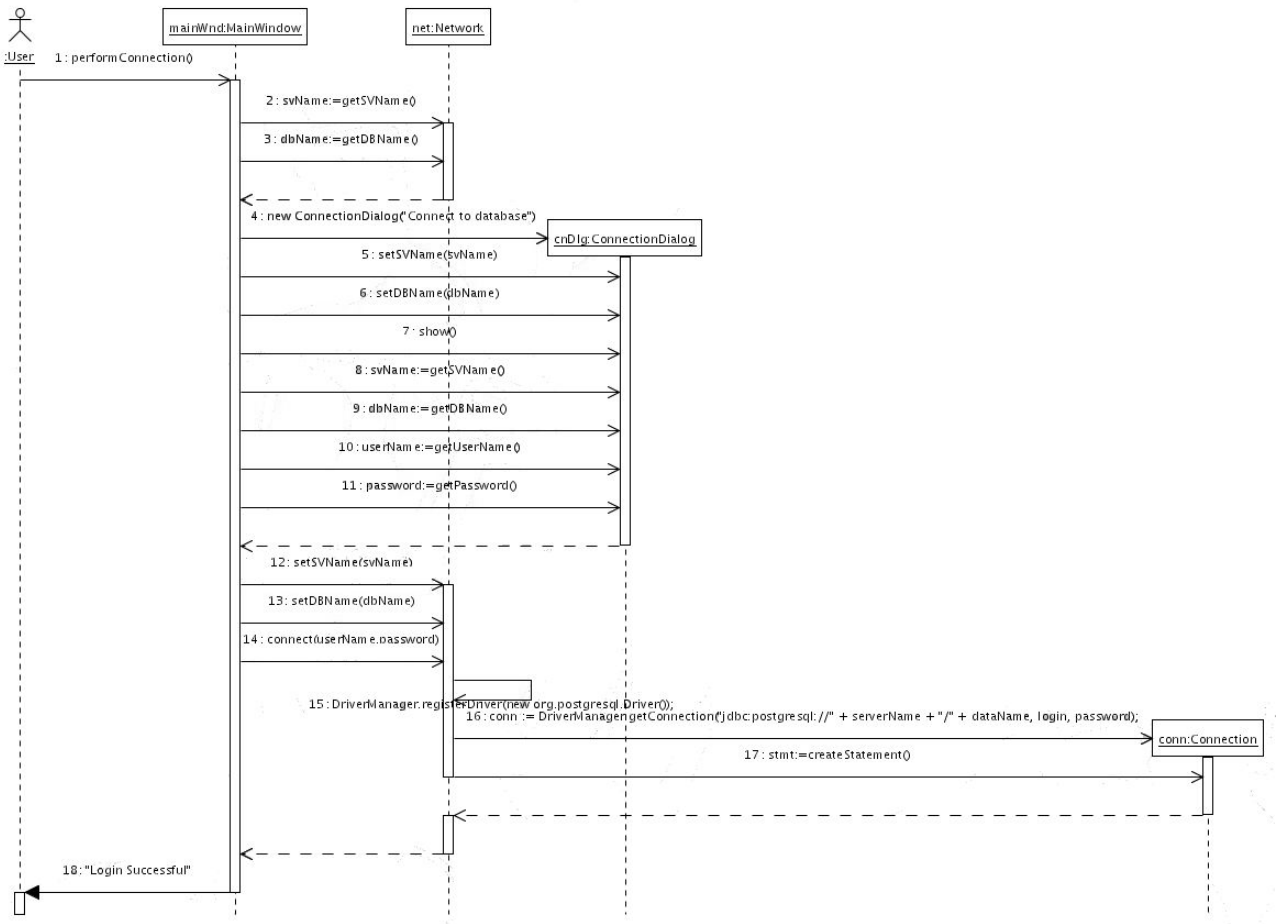


La phase de connexion

Pour la phase de connexion, nous avons donc décidé d'utiliser une fenêtre qui demanderait à l'utilisateur les informations nécessaires. C'est cette fenêtre qui est donc nommée ConnectionDialog (Boîte de Dialogue de Connexion).

La classe MainWindow, Interface Homme-Machine, est donc chargée d'appeler cette fenêtre et de demander ensuite la connexion au noyau.

Le diagramme de séquence de cette action se présente comme suit:



On voit donc que toutes les informations concernant la connexion doivent être récupérées, puis transmises à la classe principale, qui elle établit une connexion avec la base de données par le biais de la classe Connection.

Une fois cette connexion effectuée, on avertit donc l'utilisateur que la connexion s'est déroulée sans problème, par le message « Login successful ».

La phase de création

La phase de création est la plus longue du programme, elle consiste donc en une succession de demandes à l'utilisateur sur ses différents choix. Un dialogue constant entre le noyau et l'interface est donc nécessaire.

En premier lieu, le système vérifie que l'on est bien connecté à une base de données. Si ce n'est pas le cas, on le fait savoir à l'utilisateur et on n'ira pas plus loin.

Sinon, on remet à zéro le réseau, puis on demande à l'utilisateur quel mode de sélection il désire utiliser (entre les modes par taxon (Taxonomie), par organisme ou sélection de toutes les molécules sans distinction de localisation).

On va détailler distinctement les 3 choix différents possibles, même si une grande partie des étapes de ceux-ci sont identiques.

Sélection par taxon

Une fois ce choix fait, on informe le noyau du choix de la sélection par taxon, qui doit être sauvegardé pour plus tard. On demande alors à la base la liste des différents taxons disponibles, et on en offre le choix à l'utilisateur par l'intermédiaire de la fonction « viewTaxonomySelectionDialog ».

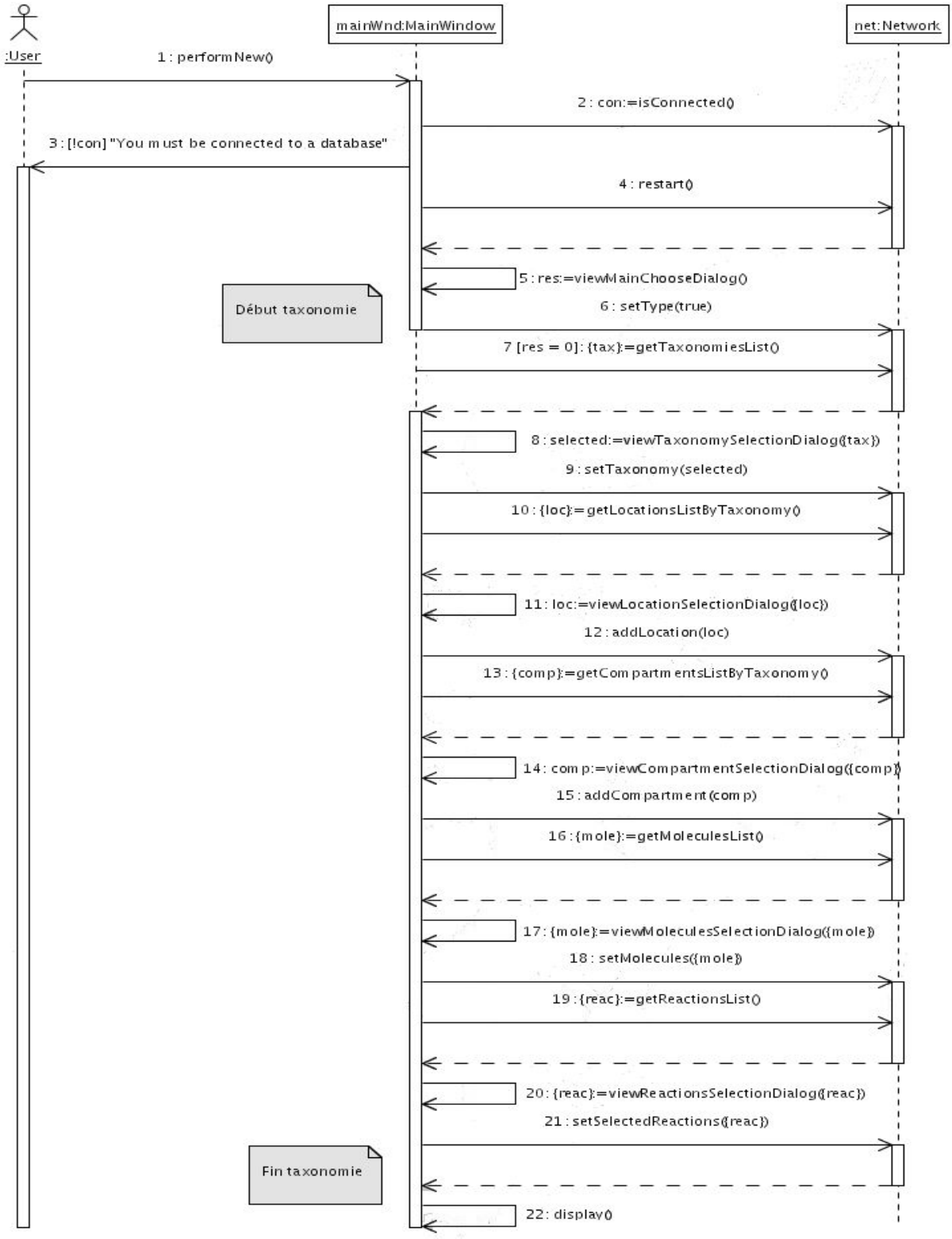
Après ce choix, on informe le noyau du taxon choisi par l'utilisateur, et le système se renseigne sur la liste des différentes locations disponibles dans ce taxon. On demande donc à l'utilisateur de choisir parmi une de ces locations par l'intermédiaire de la fonction « viewLocationSelectionDialog ».

On passe ensuite au choix du compartiment, que le système trouve dans la base de données à la fois grâce à la location et au taxon. On propose donc à l'utilisateur un choix de compartiment par l'intermédiaire de la fonction « viewCompartmentSelectionDialog ».

Le compartiment est donc transmis au noyau, et le système récupère donc la liste des molécules, et demande donc à l'utilisateur d'en choisir une ou plusieurs. Une fois ce choix fait par l'intermédiaire de la fonction « viewMoleculesSelectionDialog », les molécules sont ajoutées au compartiment actif et une liste des réactions possibles avec ces molécules est demandée à la base de données.

On demande à l'utilisateur de choisir une ou plusieurs réactions grâce à la fonction « viewReactionsSelectionDialog », et on récupère cette liste pour la sauvegarder dans le noyau, et pour terminer on affiche les informations différentes avec la fonction « display ».

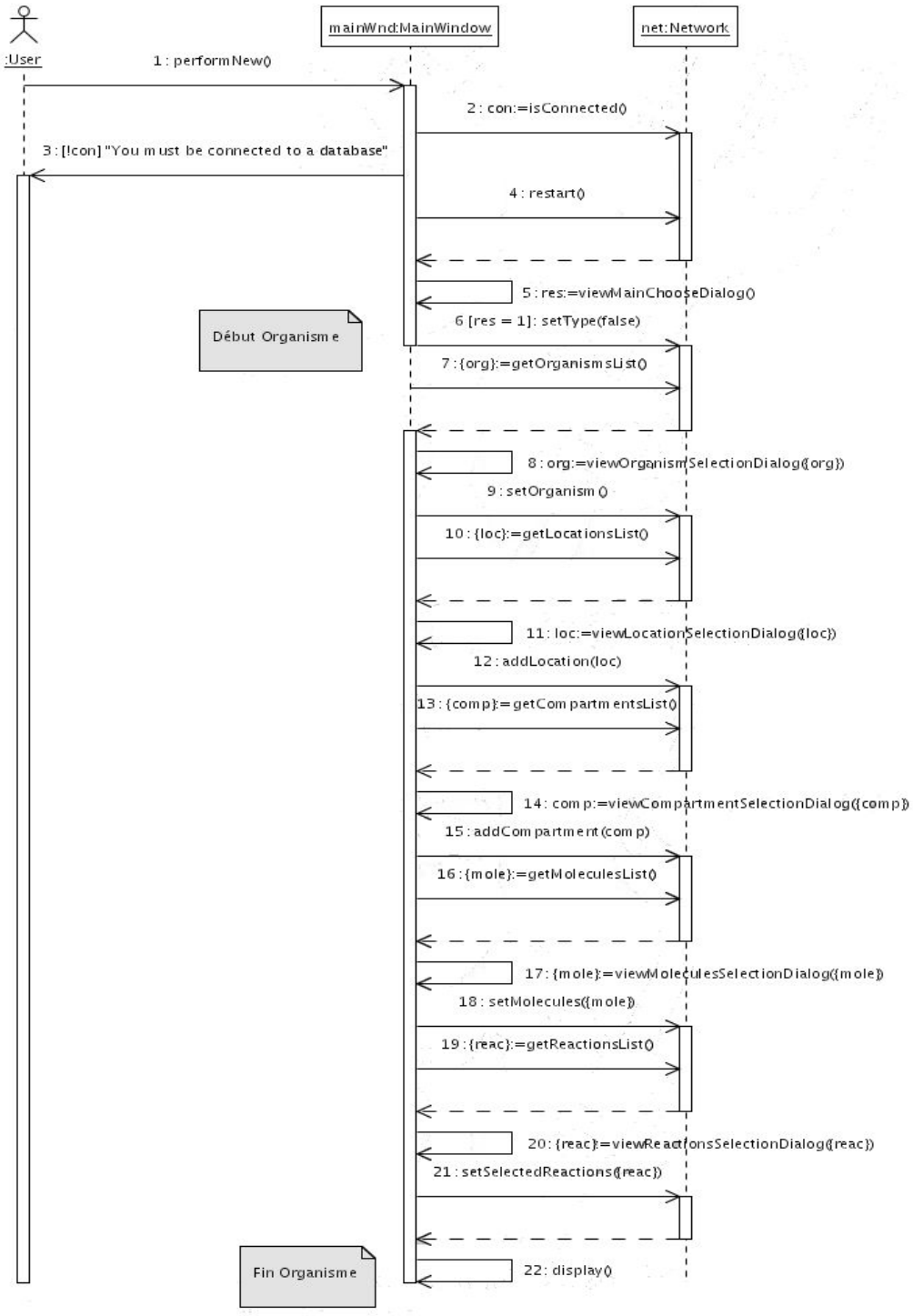
Le diagramme de séquence de la sélection par taxon se présente donc comme ceci:


:User
mainWnd:MainWindow
net:Network
1 : performNew()
2 : con:=isConnected()
3 : [!con] "You must be connected to a database"
4 : restart()
5 : res:=viewMainChooseDialog()
Début taxonomie
6 : setType(true)
7 [res = 0]: {tax}:=getTaxonomiesList()
8 : selected:=viewTaxonomySelectionDialog({tax})
9 : setTaxonomy(selected)
10 : {loc}:=getLocationsListByTaxonomy()
11 : loc:=viewLocationSelectionDialog({loc})
12 : addLocation(loc)
13 : {comp}:=getCompartmentsListByTaxonomy()
14 : comp:=viewCompartmentSelectionDialog({comp})
15 : addCompartment(comp)
16 : {mole}:=getMoleculesList()
17 : {mole}:=viewMoleculesSelectionDialog({mole})
18 : setMolecules({mole})
19 : {reac}:=getReactionsList()
20 : {reac}:=viewReactionsSelectionDialog({reac})
21 : setSelectedReactions({reac})
Fin taxonomie
22: display()

Sélection par organisme

La sélection par organisme commence par demander la liste des organismes présents dans la base au noyau, puis à faire choisir à l'utilisateur un organisme par l'intermédiaire de la fonction « viewOrganismSelectionDialog ». Le reste se déroule comme dans la sélection par taxon.

Le diagramme de séquence pour la sélection des organismes ce présent comme suit :

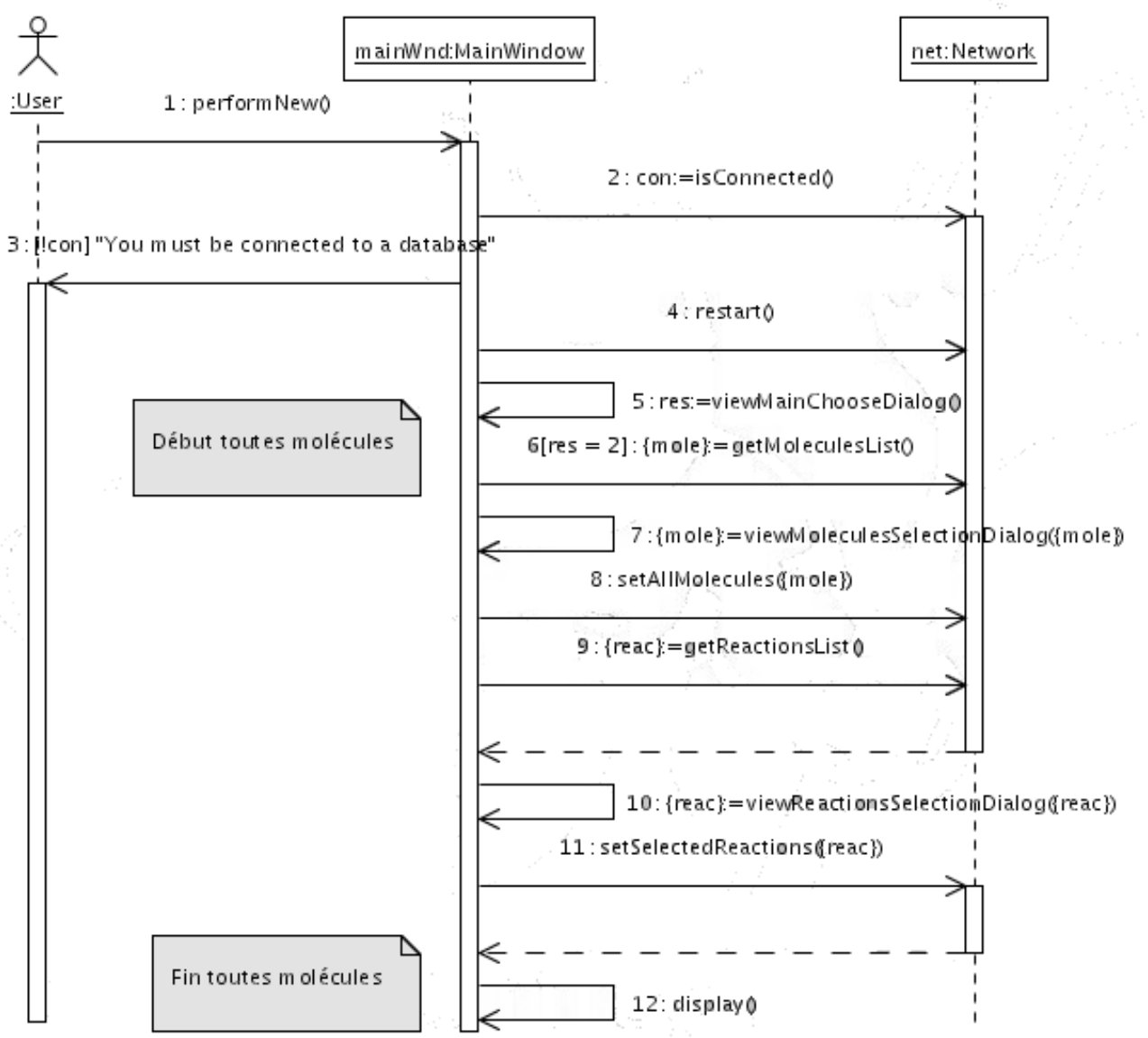


Sélection par toutes les molécules sans distinction de localisation

Cette possibilité est plus courte que les autres, étant donné qu'il n'y a pas à passer par les étapes de sélection de taxon/organisme, location et compartiment.

On a donc juste les étapes de sélection de molécules et réactions, identiques aux cas précédents.

Le diagramme de séquence se présente donc comme suit :



Phase de complétion

La phase de complétion consiste à ajouter des molécules et réactions à la liste. On ne peut y accéder que si on a fait au départ une sélection par organisme ou taxon.

Les deux possibilités différentes sont très proches, ne variant que pour deux fonctions, et donc on va décrire le fonctionnement général des deux et ensuite présenter les diagrammes de chaque possibilités.

Tout d'abord, on demande au noyau quel type de sélection a été utilisé grâce à la fonction « getType », qui retourne une valeur booléenne (VRAI si on a utilisé le mode de sélection par taxon, FAUX sinon).

Ensuite, on demande au noyau la liste des locations disponibles dans le taxon ou organisme, et on fait sélectionner à l'utilisateur une location. Puis on réitère la même opération pour les compartiments, à l'exception près que les compartiments de la location ayant déjà été sélectionnés dans cette même location ne pourront pas être sélectionnés à nouveau.

On passe ensuite à la sélection des molécules dans ce compartiment, puis à la sélection des réactions, toujours avec les mêmes fonctions utilisées dans la création.

Enfin, on réaffiche toute la fenêtre principale avec toutes les molécules et réactions.

Les diagrammes de séquence sont donc faits pour les deux possibilités de sélection.

Diagramme de séquence en cas de sélection par taxon :

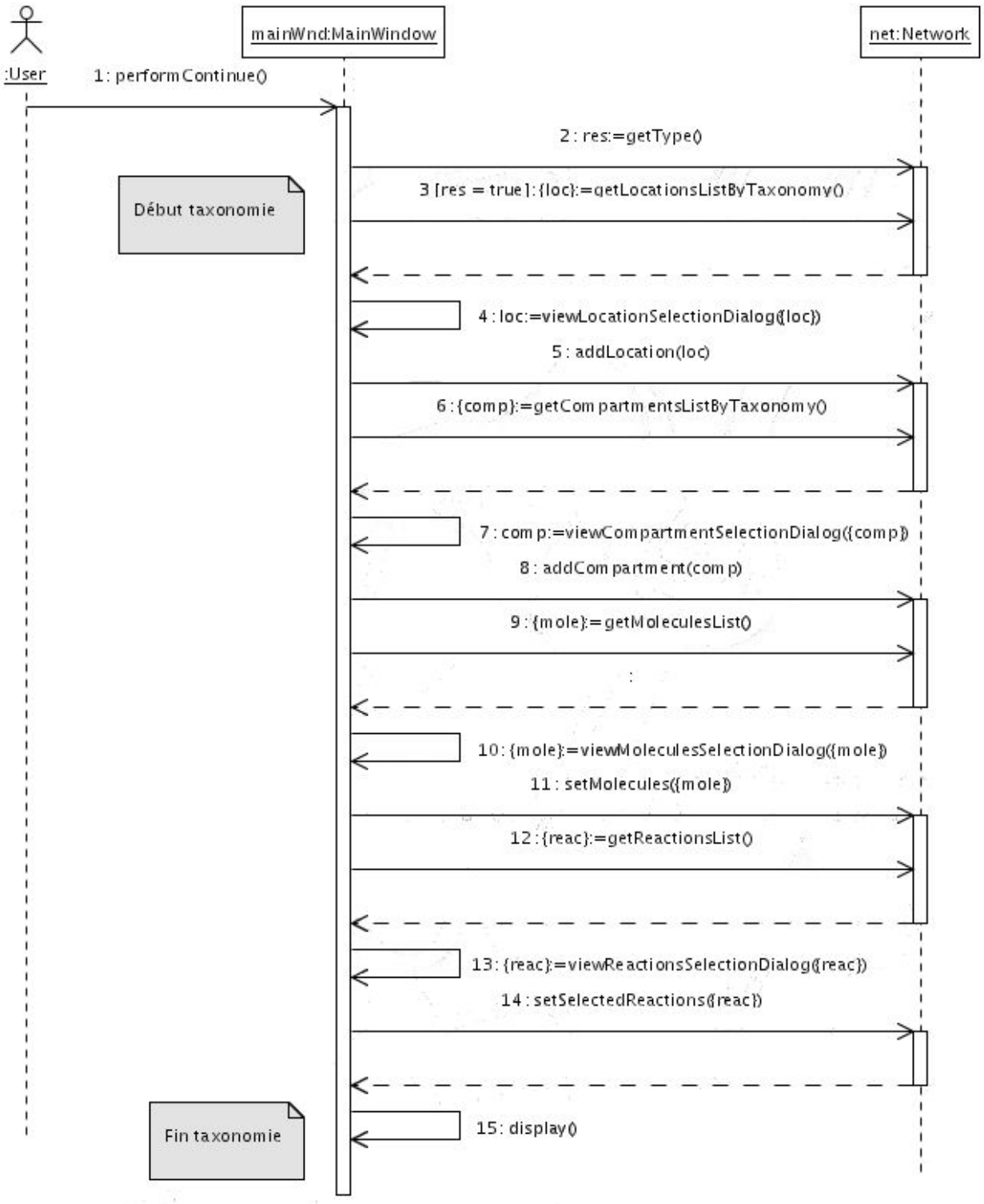
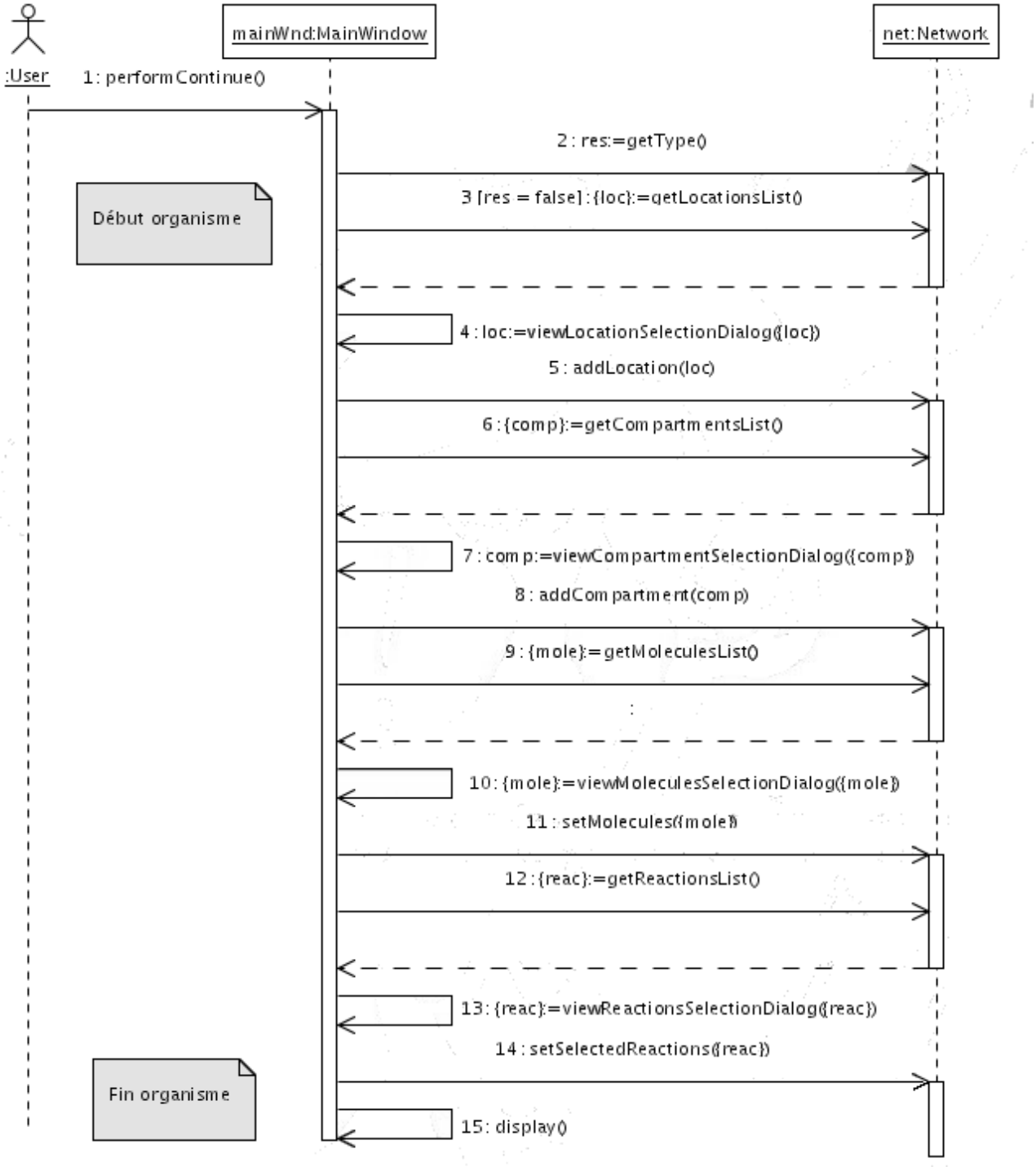


Diagramme de séquence en cas de sélection par organisme :



Phase de finalisation

Pour cette phase, il n'a pas été possible de réaliser une conception propre, étant donné qu'il s'agit surtout de recopier les molécules et réaction et permettre leur modification.

La présence de l'interface graphique rend tout dialogue impossible à représenter en diagramme pour cette étape, on n'en présente donc pas.

Phase de sauvegarde

Ici, il s'agit donc de sauvegarder les travaux effectués sur le réseau dans un fichier de type SBML.

On va donc en premier lieu demander à l'utilisateur de rentrer le nom du fichier dans lequel il souhaite sauvegarder son travail.

Puis on sauvera tout le réseau dans ce fichier par l'intermédiaire de la fonction « saveToSBMLFile ».

Le diagramme de séquence de la phase de sauvegarde se présente comme suit :

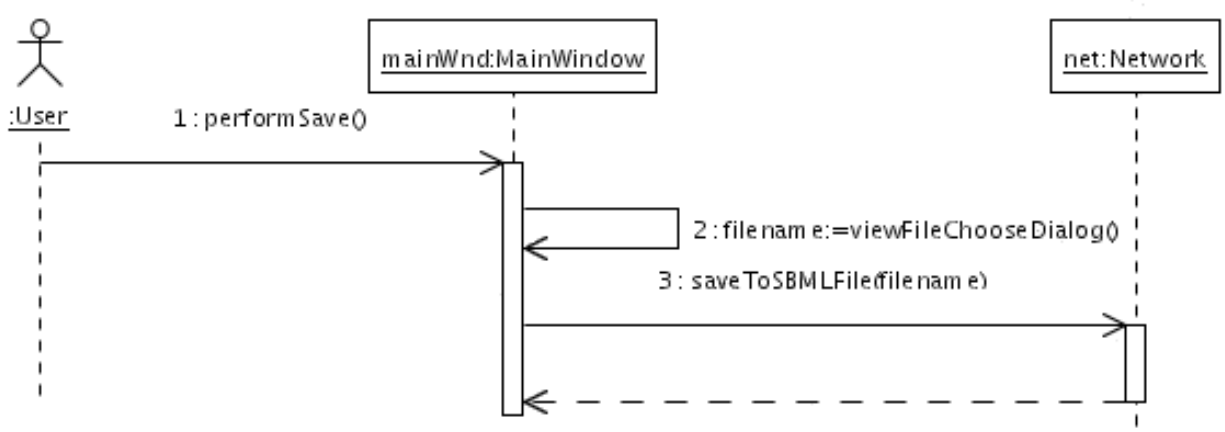


Diagramme de classes

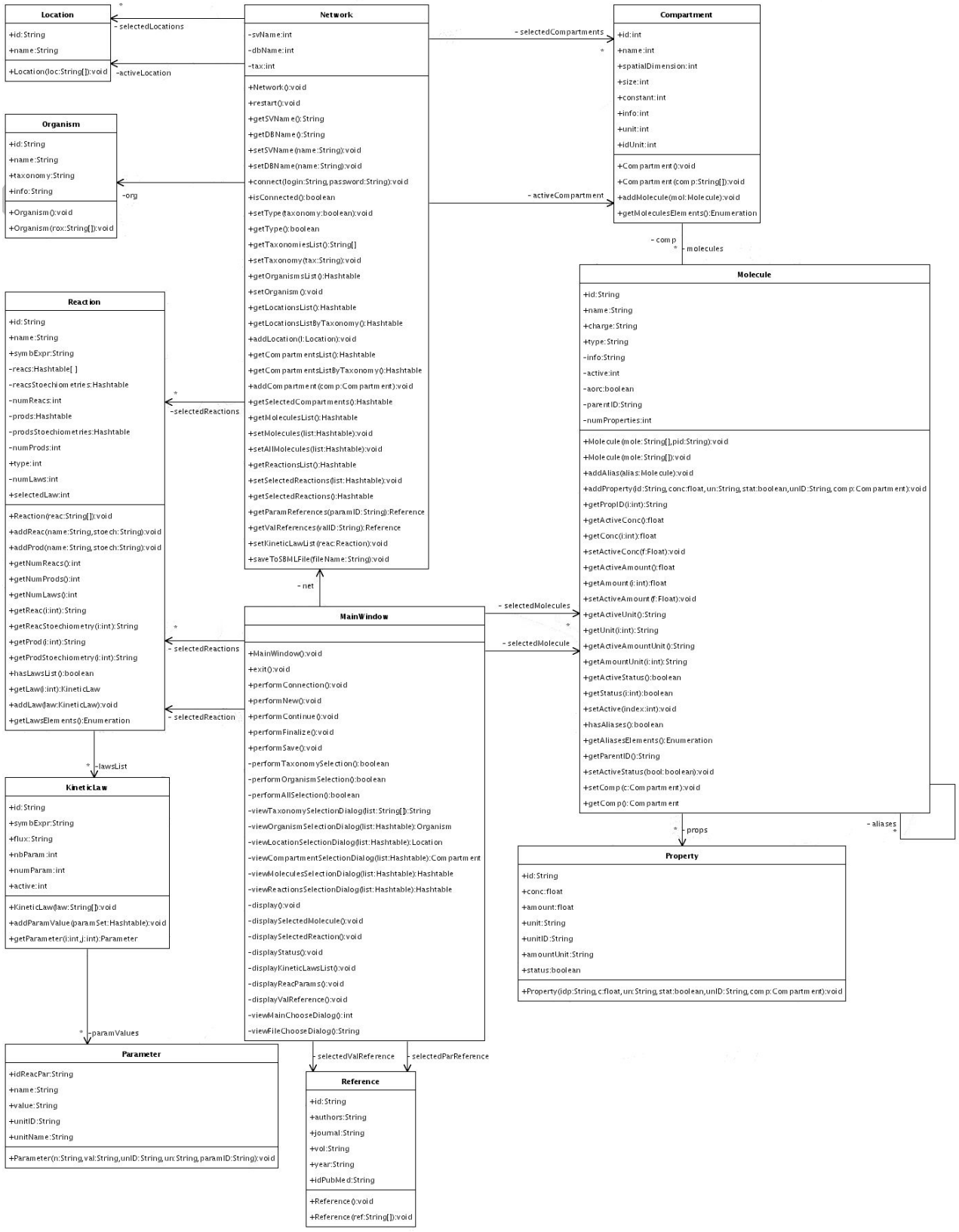
Nous avons donc, après avoir considéré les différentes actions nécessaires, pu établir une hiérarchie de classes. Outre les deux classes MainWindow et Network, principaux éléments du programme, nous avons crée une classe pour chaque type d'objet ayant à être lu dans la base de données, pour pouvoir le stocker sans consulter plusieurs fois la base de données.

Il y a donc des classes pour :

- Les organismes : « Organism »
- Les locations : « Location »
- Les compartiments : « Compartment »
- Les molécules : « Molecule »
- Les propriétés de molécules : « Property »
- Les réactions : « Reaction »
- Les lois cinétiques associées aux réactions : « KineticLaw »
- Les paramètres de lois cinétiques : « Parameter »
- Les références de provenances des différentes valeurs : « Reference »

Les classes relatives aux boîtes de dialogue ne sont pas représentées, pour des raisons de visibilité.

Le diagramme de classes est donc présenté page suivante :


Location
+id:String
+name:String
+Location(loc:String[]):void
Network
-svName:int
-dbName:int
-tax:int
+Network():void
+restart():void
+getSVName():String
+getDBName():String
+setSVName(name:String):void
+setDBName(name:String):void
+connect(login:String,password:String):void
+isConnected():boolean
+setType(taxonomy:boolean):void
+getType():boolean
+getTaxonomiesList():String[]
+setTaxonomy(tax:String):void
+getOrganismsList():Hashtable
+setOrganism():void
+getLocationsList():Hashtable
+getLocationsListByTaxonomy():Hashtable
+addLocation(l:Location):void
+getCompartmentsList():Hashtable
+getCompartmentsListByTaxonomy():Hashtable
+addCompartment(comp:Compartment):void
+getSelectedCompartments():Hashtable
+getMoleculesList():Hashtable
+setMolecules(list:Hashtable):void
+setAllMolecules(list:Hashtable):void
+getReactionsList():Hashtable
+setSelectedReactions(list:Hashtable):void
+getSelectedReactions():Hashtable
+getParamReferences(paramID:String):Reference
+getValReferences(valID:String):Reference
+setKineticLawList(reac:Reaction):void
+saveToSBMLFile(fileName:String):void
Compartment
+id:int
+name:int
+spatialDimension:int
+size:int
+constant:int
+info:int
+unit:int
+idUnit:int
+Compartment():void
+Compartment(comp:String[]):void
+addMolecule(mol:Molecule):void
+getMoleculesElements():Enumeration
Organism
+id:String
+name:String
+taxonomy:String
+info:String
+Organism():void
+Organism(rox:String[]):void
Reaction
+id:String
+name:String
+symbExpr:String
-reacs:Hashtable[ ]
-reacsStoechiometries:Hashtable
-numReacs:int
-prods:Hashtable
-prodsStoechiometries:Hashtable
-numProds:int
+type:int
-numLaws:int
+selectedLaw:int
+Reaction(reac:String[]):void
+addReac(name:String,stoech:String):void
+addProd(name:String,stoech:String):void
+getNumReacs():int
+getNumProds():int
+getNumLaws():int
+getReac(i:int):String
+getReacStoechiometry(i:int):String
+getProd(i:int):String
+getProdStoechiometry(i:int):String
+hasLawsList():boolean
+getLaw(i:int):KineticLaw
+addLaw(law:KineticLaw):void
+getLawsElements():Enumeration
Molecule
+id:String
+name:String
+charge:String
+type:String
-info:String
-active:int
-aorc:boolean
-parentID:String
-numProperties:int
+Molecule(mole:String[],pid:String):void
+Molecule(mole:String[]):void
+addAlias(alias:Molecule):void
+addProperty(id:String,conc:float,un:String,stat:boolean,unID:String,comp:Compartment):void
+getPropID(i:int):String
+getActiveConc():float
+getConc(i:int):float
+setActiveConc(f:Float):void
+getActiveAmount():float
+getAmount(i:int):float
+setActiveAmount(f:Float):void
+getActiveUnit():String
+getUnit(i:int):String
+getActiveAmountUnit():String
+getAmountUnit(i:int):String
+getActiveStatus():boolean
+getStatus(i:int):boolean
+setActive(index:int):void
+hasAliases():boolean
+getAliasesElements():Enumeration
+getParentID():String
+setActiveStatus(bool:boolean):void
+setComp(c:Compartment):void
+getComp():Compartment
MainWindow
+MainWindow():void
+exit():void
+performConnection():void
+performNew():void
+performContinue():void
+performFinalize():void
+performSave():void
-performTaxonomySelection():boolean
-performOrganismSelection():boolean
-performAllSelection():boolean
-viewTaxonomySelectionDialog(list:String[]):String
-viewOrganismSelectionDialog(list:Hashtable):Organism
-viewLocationSelectionDialog(list:Hashtable):Location
-viewCompartmentSelectionDialog(list:Hashtable):Compartment
-viewMoleculesSelectionDialog(list:Hashtable):Hashtable
-viewReactionsSelectionDialog(list:Hashtable):Hashtable
-display():void
-displaySelectedMolecule():void
-displaySelectedReaction():void
-displayStatus():void
-displayKineticLawsList():void
-displayReacParams():void
-displayValReference():void
-viewMainChooseDialog():int
-viewFileChooseDialog():String
KineticLaw
+id:String
+symbExpr:String
+flux:String
+nbParam:int
+numParam:int
+active:int
+KineticLaw(law:String[]):void
+addParamValue(paramSet:Hashtable):void
+getParameter(i:int,j:int):Parameter
Property
+id:String
+conc:float
+amount:float
+unit:String
+unitID:String
+amountUnit:String
+status:boolean
+Property(idp:String,c:float,un:String,stat:boolean,unID:String,comp:Compartment):void
Parameter
+idReacPar:String
+name:String
+value:String
+unitID:String
+unitName:String
+Parameter(n:String,val:String,unID:String,un:String,paramID:String):void
Reference
+id:String
+authors:String
+journal:String
+vol:String
+year:String
+idPubMed:String
+Reference():void
+Reference(ref:String[]):void
- selectedLocations
- activeLocation
- selectedCompartments
- activeCompartment
- org
- comp
- molecules
- selectedReactions
- net
- selectedMolecules
- selectedMolecule
- selectedReaction
- lawsList
- paramValues
- selectedValReference
- selectedParReference
- aliases
- props

Réalisation

Dès le début de la réalisation, nous nous sommes rendu compte que nous ne pourrions pas coller aux étapes qui été définies a titre indicatif par Julien Berro. Nous avons donc du changer certaines étape du planning prévisionnel pour mieux coller au nécessiter du logiciel.

Comme il est visible sur le diagramme de classes, deux classes se démarque: la classe MainWindow et la classe Network. En gros, ces classes nous ont permis une répartitions du travail claire, la classe Network étant le noyau applicatif du logiciel, et la classe MainWindow étant la classe gérant l'interface Homme-Machine. Le travail fut donc répartit en deux parties, Benoît Clément travaillant sur la partie interface Homme-Machine et Benjamin Millau travaillant sur la partie Noyau Applicatif (typiquement, tous ce qui se trouve dans la classe Network).

Pour les autres classes représentant tous les éléments présent dans la base et nécessaire pour le logiciel comme Molecule, ou Reaction, leur développement a été effectué au fur et a mesure des besoins, par Benoît ou par moi-même tout au long du développement du logiciel.

Etapes de réalisation

Voici les étapes du développement pour le noyau applicatif. Plusieurs étapes ne concernant que l'interface Homme-Machine ne sont pas mentionnées ici.

Etape 1 : installation et configuration.

Comme dans le planning prévisionnel, avant de commencer la réalisation, il était nécessaire de posséder tous les logiciels utiles pour cette réalisation. Etant donnée que nous avions choisi de programmer en Java, nous avons choisi Jbuilder comme logiciel de développement. Il a fallut donc l'installer sur les deux postes, et le configurer correctement. Une librairie permettant un dialogue entre la base de données PostgreSQL et le logiciel Java était nécessaire, la librairie JBDC a donc été choisie et configurée.

Ensuite, nous avons du installer et configurer CVS, nous permettant ainsi de pouvoir mettre en commun tout le travail réalisé chacun sur notre poste. Le serveur CVS, mis en place par Julien Berro, est physiquement logée sur une autre machine de TIMC, il est donc nécessaire de gérer les identités pour la connexion. Grâce à une interface de type tkCVS, la configuration a pu se faire sans trop de problèmes, et ainsi, après avoir compris le fonctionnement de CVS, nous avons pu commencer la réalisation du logiciel à proprement parler.

Etape 2 : connexion à une base de données.

La phase permettant à l'utilisateur de se connecter à la base de données.

Etape 3 : choix des molécules.

Toutes les informations allant du choix des modes à la sélection des molécules.

Etape 4 : choix des réactions.

Implémentation de toutes les méthodes permettant la sélection des réactions ainsi que celle qui permettent d'obtenir toutes les informations sur ces réactions, comme les lois cinétiques et leurs paramètres.

Etape 5 : prise en compte du rajout d'élément.

Modification de certaines méthodes, afin que celles ci puissent prendre en compte les rajouts de compartiments, de locations et de molécules.

La classe Network

Phase de connexion.

La première étape du logiciel était évidemment la phase de connexion à la base de données. L'interface et le noyau ont été développés conjointement, permettant donc de pouvoir tester directement les méthodes écrites.
Pour pouvoir utiliser la librairie JDBC, il faut d'abord importer la librairie de cette façon:

```
import java.sql.*;
```

Ensuite, pour exécuter la connexion, une méthode dans la classe Network a donc du être créée.

```
public boolean connect(String login, String password)
```

Cette méthode renvoie un booléen qui permet de confirmer si la connexion a bien été établie. Seulement 2 paramètres « String » correspondant au nom et au mot de passe sont demandés car les autres informations sont stockées dans un fichier « serveur.ini » qui sera abordé plus loin dans cette partie.
Cette méthode est très importante, car c'est elle qui déclare le pilote de la base de données et qui se connecte à cette base. Tout ceci ce fait grâce à JDBC.
La déclaration du pilote se fait ainsi :

```
DriverManager.registerDriver(new org.postgresql.Driver());
```

Ainsi, le pilote est initialisé et prêt à être utilisé pour la connexion. Ensuite, la connexion est possible grâce à une méthode. L'utilisation se fait ainsi:

```
conn = DriverManager.getConnection("jdbc:postgresql://" + serverName + "/" +
          dataName, login, password);
```

Pour se connecter à la base de données, il n'y a besoin que d'une chaîne de caractère doit contenir le nom du serveur où se situe la base de données physiquement, le nom cette base de données, le nom de l'utilisateur, et le mot de passe d'accès a la base de données.
Ensuite, une instance de la connexion « conn » doit être effectuée. Elle s'effectue de cette façon:

```
stmt = conn.createStatement();
```

Cette instance est importante, car elle va permettre de pouvoir effectuer toutes les requêtes SQL que le logiciel aura besoin.
Comme dit précédemment, le nom du serveur et le nom de la base de données sont écrit dans un fichier « serveur.ini ». Ce fichier est lu à chaque ouverture du logiciel. Cela permet aux utilisateurs de ne pas avoir à les réécrire à chaque ouverture du logiciel.

Choix du mode.

Après s'être connecté à la base de données, l'utilisateur a le choix entre trois modes de sélection. Une fenêtre de dialogue s'ouvre alors pour permettre à l'utilisateur de faire son choix. Une fois le choix effectué, la classe « MainWindow » appelle la méthode suivante :

```
public void setType(boolean taxonomy)
```

Cette méthode initialise la variable de la classe Network « type » à « true » si l'utilisateur a choisi de travailler à partir de taxon, ou à « false » si l'utilisateur a choisi de travailler à partir des organismes.
Cela sera utile plus tard pour la classe « MainWindow », afin de faire différents tests.

Choix du taxon.

Viennent ensuite les méthodes permettant d'afficher la liste des organismes ou des taxons. Ces méthodes sont similaires, mis à part bien sûr la requête qui permet le dialogue avec la base de données. Dans un premier temps, nous allons voir la requête permettant d'obtenir la liste des taxons afin de pouvoir afficher cette liste dans une fenêtre. Cette méthode est la suivante :

```
public String[] getTaxonomiesList()
```

Cette méthode représente typiquement les méthodes qui permettent un dialogue entre le logiciel et la base de données à l'aide de JBDC. Nous allons voir comment utiliser JBDC.
Tout d'abord, il faut déclaré une variable de type « ResultSet » :

```
ResultSet rs;
```

Le « ResultSet » est en quelque sorte un tableau à deux dimensions. Cette variable contiendra le résultat de la requête SQL. La requête s'effectue ainsi :

```
rs = stmt.executeQuery("SELECT DISTINCT taxonomy FROM organism");
```

Cette instruction appelle une autre méthode « executeQuery(String) » qui se trouve dans le paquetage « stmt ». C'est donc pour cela que, lors de la connexion, il est nécessaire d'instancier la connexion « conn ». Cette méthode permet donc d'exécuter la requête SQL « SELECT DISTINCT taxonomy FROM organism », requête qui prend donc la liste des taxons à partir de la table « organism ». Le résultat de la méthode, et donc de la requête, est stocké dans la variable « rs ».
Ensuite, afin que les informations soient facilement utilisables, il est plus simple de mettre toutes les informations contenues dans la variable « rs » dans une autre variable de type « String[] ». Afin d'initialiser la variable « taxonomies » de type « String[] », il est nécessaire tout d'abord de compter le nombre de résultat contenu dans la variable « rs ». Pour cela, une méthode « next() » du paquetage « ResultSet » permet de passer au résultat suivant et donc pour compter les résultat, il suffit de faire :

```
while (rs.next()) {
            i++;
}
```

La variable « i » étant de type « integer ». Elle contient le nombre de résultat contenu dans la variable « rs ». Nous pouvons ensuite déclaré la variable taxonomies de cette façon :

```
taxonomies = new String[i];
```

Ensuite, afin de pouvoir mettre toutes les informations dans la variable « taxonomies », il suffit de procéder de cette façon :

```
rs.first();
int j = 0;
do {
 taxonomies[j] = rs.getString(1);
}
while (rs.next());
```

La méthode « first() » du paquetage « ResultSet » permet de retourner directement au premier élément du « ResultSet ». Puis la méthode « getString(int i) » permet d'obtenir le résultat sous forme de « String », la variable « i » indiquant quelle colonne de la ligne courante est voulue, qui va être enregistré dans une des positions du tableau « taxonomies ». Puis, on appelle la méthode « next() » afin de passer au résultat suivant.
Enfin, il ne reste plus qu'à retourner le tableau « taxonomies ».

Choix de l'organisme.

Si l'utilisateur a choisi de sélectionner ses molécules à partir d'organismes, la fenêtre principale va faire un appel à la méthode suivante:

```
public Hashtable getOrganismsList()
```

Cette méthode, va récupérer la liste de tous les organismes contenus dans la base de données, et va les retourner dans une variable de type « Hashtable », qui est en faite une liste indexée, qui va contenir des élément de type « Organism ». Si on veut mettre les informations dans un fichier de type « Hashtable », c'est parce que cette fois si, il y a plus d'élément à sauvegarder qu'une simple liste de nom de taxons. La requête correspondant s'écrit donc ainsi:

```
SELECT * FROM organism;
```

Ainsi, il y aura plusieurs éléments à enregistrer dans la « Hashtable », nous avons donc procédé ainsi:

```
Rs.first();
Do {
  For (int k = 0 ; k < numOrganismTitles; k++){
    Orgas[j][k] = rs.getString(k + 1);
  }
  J++;
 }
While (rs.next());
  Result = new Organism[i];
  For(i = 0; i < numOrganismRows; i++){
    Result[i] = new Organism(orgas[i]);
  }
Hashtable res = new Hashtable();
For (i = 0; i < result.length; i++){
  Res.put(result[i].id, result[i]);
}
Return res;
}
```

Pour bien comprendre le code, il faut savoir que « numOrganismTitles » est un attribut de la classe Network qui permet de savoir combien de champ la table « organism » contient. Ensuite, « orgas[i][j] » a été déclaré comme un « String[][] », « result » a été déclaré comme un tableau d'élément de type « Organism » et « numOrganismsRows » contient le nombre de résultat dans le « ResultSet », qui a été trouvé juste avant de la même façon que dans la méthode « getTaxonomiesList() ».
Pour cette partie, on enregistre tout d'abord tout les résultats contenus dans le « ResultSet » dans une variable de type « String[][] ». Puis, les chaînes de caractères contenues dans ce tableau de « String » vont être entrés dans un autre tableau d'élément de type « Organism ». Pour cela, nous prenons tout une ligne du tableau « orgas », et grâce au constructeur de la classe « Organism », il est possible de créer un tableau « result ». Pour bien comprendre le fonctionnement, il est nécessaire de voir le constructeur de la classe « Organism ». Il a été écrit de cette façon:

```
public Organism(String []
row) {
   id = row[0];
   name = row[1];
    taxonomy = row[2];
    info = row[3];
  }
```

C'est donc pour cela que l'instruction : « result[i] = new Organism(orgas[i]); » est possible. Ensuite, tout le tableau « result » de « Organism » est entré dans une variable de type « Hashtable », indexé sur l' « id » de l'organisme, puisque « id » est la clef de cette table dans la base de données et ainsi que dans le logiciel.
Ensuite, pour le bon fonctionnement rapide et simple du logiciel, le taxon ou l'organisme choisi par l'utilisateur doit être gardé dans une variable du type approprié.
Par exemple pour un organisme sélectionné, on va le garder dans une variable afin de le réutiliser plus tard. Cette variable se déclare ainsi:

```
private Organism org;
```

Pour cela, lors de la validation du choix par l'utilisateur, la fenêtre « MainWindow » appelle cette méthode:

```
public void setOrganism(Organism o) {
org = o;
}
```

Cette méthode initialise tout simplement la variable « org » avec l'organisme entré en paramètre, c'est à dire celui choisi par l'utilisateur. Dans le cas du taxon, cela se passe exactement de la même façon, avec une variable et une méthode différente bien sûr.

Choix de la location.

Ensuite, après avoir choisi un taxon ou un organisme, la liste des locations contenues dans ce taxon ou cet organisme doit être récupéré. Pour cela, selon le choix effectué, la fenêtre « MainWindow » appelle une des deux méthodes de la classe Network. Soit elle appelle « public Hashtable getLocationsList() » qui permet d'obtenir la liste des locations à partir d'un organisme, soit elle appelle « public Hashtable getLocationsListByTaxonomy() » qui permet d'obtenir la liste des locations à partir d'un taxon choisi.
Ces méthodes sont semblable à la méthode « getOrganismsList », à la différence de quelques variables et méthodes appelées, mais la grande différence se situe au niveau de la requête SQL, qui cette fois ci va s'écrire ainsi:

```
"SELECT  loc.id, loc.name
            FROM  organism org, location loc, reac_param_location rpl
            WHERE  org.id = rpl.id_organism  AND  loc.id = rpl.id_location AND
            org.id = '" + org.id + "'";
```

Cette requête SQL est plus spéciale que les autres, puisqu'elle fait intervenir des attributs propre au Java. Ceci se fait de manière simple, puisque la requête SQL pour Java n'est qu'une chaîne de caractères, il est donc possible de concaténer n'importe quel autre chaîne de caractères. Et c'est ce qui est fait ici, « org.id » étant un « String », cela fonctionne parfaitement.
La méthode « getLocationsListByTaxonomy() » fait exactement la même chose, mais à partir d'un taxon connu.

Comme pour l'organisme ou le taxon il est nécessaire de garder la location choisie dans une variable. Mais, étant donnée que l'on pourra choisir plus tard une autre location, nous avons défini un attribut de type « Hashtable » contenant toutes les locations sélectionnées par l'utilisateur, en plus d'une variable contenant la location actuellement active.

```
private Hashtable selectedLocations = new Hashtable();
          private Location activeLocation;
```

Donc une méthode existe aussi pour pouvoir initialiser puis rajouter des locations. Cette méthode est:

```
public void addLocation(Location l)
```

Cette méthodes est simple: elle rajoute la « location l » dans la variable « selectedLocations », en l'indexant sur son « id ». Puis elle initialise la variable « activeLocation » par la « location l ».

Choix du compartiment.

Après ce choix effectué, l'utilisateur aura un nouveau choix à faire : celui du compartiment. La liste de tous les compartiments disponibles dans la location choisie précédemment doit s'afficher.
Pour cela, la fenêtre « MainWindow » appelle de nouveau une autre méthode de la classe « Network ».
Là encore, il y a deux méthodes qui font la même chose à la différence près qu'une construit la liste à l'aide du taxon sélectionné, et l'autre à partir de l'organisme sélectionné. Ces méthodes sont:

```
public Hashtable getCompartmentsList(Location loc)
public Hashtable getCompartmentsListByTaxonomies(Location loc)
```

Ces méthodes sont intéressantes, car a priori, elles devraient être semblables à « getLocationsList() », alors qu'il était demandé pour le logiciel, qu'une fois un compartiment choisi, celui ci ne devrait pas se réafficher pour éviter les erreurs. Nous avons choisi de gérer cela directement au niveau de la requête SQL car cela nous simplifiait les choses. Il a donc fallu découper la requête en plusieurs partie, et gérer les conditions. Voici la manière de procéder:

```
String query = "SELECT cp.id, cp.name, cp.spatialdimension, cp.size,
cp.constant, cp.info, cp.id_unit, un.name FROM compartment cp, unit un WHERE
cp.id_location = '" + loc.id + "' AND cp.id_organism = '" + org.id + "' AND
un.id = cp.id_unit";
     int i = 0;
     if (!selectedCompartments.isEmpty()) {
        query += " AND cp.id NOT IN ( ";
        Enumeration enum = selectedCompartments.elements();
        while (enum.hasMoreElements()) {
            if (i > 0) {
                query += " , ";
            }
            query += " '" + ( (Compartment) enum.nextElement()).id + "' ";
            i++;
        }
        query += " ) ";
     }
```

Comme pour les locations, il y a un attribut « selectedCompartments » de type « Hashtable » qui contient tous les compartiments sélectionnés depuis le début de la création du réseau. La requête est donc décomposée ainsi: elle sélectionne dans tout les cas tout les compartiment, et ensuite, selon les compartiments contenus dans le « Hashtable selectedCompartments », elle retire tous ceux déjà présents. Après tout cela, il ne reste plus qu'à faire s'exécuter la chaîne de caractère « query » comme étant une requête SQL.
Comme pour les locations, une fois l'utilisateur ayant fait son choix, la méthode « public void addCompartment(Compartment c) » est appelée, et celle-ci rajoute le compartiment sélectionné à l'attribut « selectedCompartments » et initialise l'attribut « comp » qui représente le compartiment courant.

Choix des molécules.

Ensuite, vient le choix des molécules qui seront présent dans le compartiment choisi précédemment. Comme d'habitude, la fenêtre « MainWindow » appelle une fonction de la classe « Network » pour obtenir la liste de toutes les molécules de toute la base de données.

```
public Hashtable getMoleculesList()
```

Cet méthode ne va pas être développée car elle est tout à fait similaire au méthode « getOrganismsList() » et « getTaxonomiesList() ».

A ce moment là, l'utilisateur peut choisir plusieurs molécules en même temps, donc des méthodes de l'interface Homme-Machine vont s'occuper de récupérer les choix effectués par l'utilisateur, puis la classe « MainWindow » va appeler une autre méthode très complexe de la classe « Network ».

```
public void setMolecules(Hashtable list)
```

Cette méthode a pour objectif d'initialiser la liste des molécules choisies, c'est à dire la variable « selectedMolecules », puis récupère toutes les informations nécessaires à l'affichage, comme les quantités, les unités, et le statut de toutes les molécules de la liste. Cette méthode, est comme pour les autres basée autour d'une requête SQL, mais celle-ci est très complexe car il y a beaucoup d'éléments à prendre en compte, et ici, il faut faire une requête afin d'obtenir une liste de résultats pour plusieurs molécules dont on ne connaît pas le nombre. Il a fallu donc prévoir que la méthode fonctionne aussi bien si une molécule est choisie, ou alors un nombre indéterminé. Pour cela, il a fallut utiliser judicieusement les boucles et les conditions afin de toujours obtenir une requête valide. C'est une méthode relativement longue, vous pouvez donc la consulter en annexe (Annexe 2, page 44).
La partie la plus intéressante dans cette méthode, est sûrement la gestion du problème d'affichage des unités. Effectivement, lors de l'affichage des paramètres, certain commençant par les lettres « µm » ne fonctionnaient pas et affichaient « ? » à la place. Nous avons donc du trouver une parade à ce problème, nous avons procédé ainsi:

```
String r = rs.getString(3);
          if (Character.getNumericValue(r.charAt(0)) < 0) {
               r = "µm" + r.subSequence(1, r.length());
          }
```

L'unité se situe en troisième position dans le tableau « ResultSet », c'est pour cela que l'on récupère celui la pour le tester. Après quelques tests, nous avons remarqué que lorsque ce problème ce produit, la méthode « getNumericValue(char) » retournait « -1 ». Il nous a suffit donc de tester le premier caractère de l'unité et si celui ci est inférieur à « 0 » . Il suffit d'affecter à la variable « r » les caractères « µm » puis d'y rajouter la suite des caractères de l'unité.

Choix des réactions.

Comme toujours, la fenêtre « MainWindow » appelle une fonction dans la classe « Network ». Elle appelle cette fonction:

```
public Hashtable getReactionsList()
```

Cette fonction a pour but de créer une liste de réactions, en fonction des molécules qui ont été sélectionnées précédemment, et elle permet aussi d'obtenir les stœchiométries des réactions. C'est de loin la méthode la plus complexe et la plus longue du logiciel, elle se trouve également entièrement en annexe (Annexe 3, page 46). L'écriture de cette méthode a pris plusieurs jours avant d'arriver à ce qu'elle soit parfaitement opérationnelle.
Cette méthode est construite entièrement autour d'une requête complexe. De nombreuse variables de type « Enumeration » ont du être utilisé.
Tout d'abord, on créé la requête afin qu'elle sélectionne toutes les réactions qui ont un lien avec les molécules choisies par la table « reacts » et « products » (voir la spécification de la base de données en annexe 5 page 59). Ensuite, il est nécessaire d'éliminer les réactions qui n'ont pas que des molécules qui ont été sélectionné au préalable. C'est donc pour cela que la requête est encore complétée de la même manière que la première fois, mais avec dans la requête la condition de négation « NOT IN ». Ainsi, seules les réactions ayant les molécules sélectionnées et uniquement celles-ci seront retournées par la requête.
Ensuite, dans cette même méthode, il faut obtenir pour chaque molécule de la réaction sa stœchiométrie. Pour cela, la requête retourne le nom de la molécule et sa stœchiométrie que nous rajoutons dans le tableau contenant toutes les réactions, grâce a une méthode de la classe « Reaction ». Il faut le faire pour les molécules qui interviennent aussi bien en tant que produit ou que réactif d'une réaction.
Ensuite afin de récupérer les lois cinétiques des réactions, un appel à la méthode « setKineticLawList(Reaction reac) » est effectué.

Choix des lois cinétiques.

Enfin, la dernière méthode appellée, par « getReactionsList() » d'ailleurs, est

```
public void setKineticLawList(Reaction reac)
```

Cette méthode permet d'obtenir les lois cinétiques ainsi que les paramètres de la réaction « reac » passée en paramètre. Comme pour les autres méthodes, les requêtes SQL sont exécutées, puis les résultats sont placés dans des variables de type « Hashtable » facilement utilisable. Ainsi 3 requêtes sont classiquement exécutées et aucun problème ne se montre, excepté le même problème d'affichage que pour les unités, réglé de la même façon que précédemment.
Quelques liaisons entre les classes sont tout de même effectués comme par exemple cette instruction qui rajoute les paramètres, obtenu grâce à la requête SQL, à leur loi cinétique respective:

```
kinetics[i].addParamValue(pars);
```

Ou encore cette instruction qui ajoute à la réaction ses lois cinétiques :

```
reac.addLaw(kinetics[i]);
```

Choix des molécules sans domaine particulier de travail.

Toutes ces méthodes ne sont pas forcément appelées. Effectivement, si l'utilisateur au départ n'a pas choisi entre taxon ou organisme, alors, une autre méthode est appelée:

```
public void setAllMolecules(Hashtable list)
```

Cette méthode reçoit en paramètre toutes les molécules sélectionnées par l'utilisateur, mais étant donné que l'utilisateur n'a pas choisi ni de location, ni de compartiment, cette méthode s'occupe de trouver les valeurs possibles des molécules, et ce dans toutes les locations possible. Tout d'abord, afin de concordé avec le reste du logiciel, un compartiment factice est créé. Ainsi, une requête SQL est exécutée, permettant d'obtenir toutes les valeurs (la quantité par exemple) des molécules quelque soit la location. Puis, grâce à une méthode « addPorperty » de la classe « Molecule », les propriétés trouvées pour la molécule sont associées à cette molécule.

Obtention des références.

Enfin, pour finir avec cette classe « Network », il y a deux autres méthodes qui permettent d'obtenir les références. L'une obtient la référence à partir de la valeur de la molécule actuellement choisie, et l'autre obtient la référence à partir du jeu de paramètres de la loi cinétique actuellement sélectionnée. Ces deux méthodes sont très simples et similaires, excepté bien sûr la requête SQL qui est différente. Elle est de ce type:

```
"SELECT DISTINCT ref.id, ref.authors, ref.journal, ref.vol, ref.year_pub,
ref.id_PubMed FROM reference ref, initial_conditions ic WHERE ref.id =
ic.id_ref AND ic.id = '" + valId + "'";
```

De plus, étant donnée qu'il ne peut y avoir qu'une seule référence pour un jeu de paramètres associé à une loi cinétique ou à une valeur de molécule, il y a uniquement besoin de garder le résultat obtenu dans une variable de type « Reference », et de le retourner pour que la fenêtre principale puisse l'afficher.

Gestion des exceptions

Pour gérer les exceptions, un moyen simple et efficace a été utilisé. Toutes les exception levées dans les méthodes de la classe « Network » sont traitées de la même façon : le message de l'exception est récupéré et renvoyer a la classe qui appelle les méthode de la classe « Network », c'est-à-dire pour la plupart du temps, la classe « MainWindow », et c'est cette classe qui va s'occuper de faire afficher un message d'erreur. Pour cela, nous avons procédé de cette méthode que nous permet Java:

```
public void maMéthode() throws Exception {
   try{
      [suite d'instruction de la méthode] ;
   }
   catch (Exception e) {
      Throw new Exception(e.getMessage()) ;
   }
}
```

Les autres classes

En plus de la classe « Network », la création d'autres classes a été nécessaire pour chaque élément manipulé dans le logiciel et dans la base de données telles que les molécules, réaction, et autres références. Ces classes sont des classes simples, le constructeur de ces classes est toujours de même style, par exemple pour la classe « Organism » qui se présente ainsi:

```
public class Organism {
     public String id;
     public String name;
     public String taxonomy;
     public String info;
```

Alors le constructeur va être comme ceci:

```
public Organism(String [] row) {
     id = row[0];
     name = row[1];
     taxonomy = row[2];
     info = row[3];
}
```

En fait, ce constructeur reçoit une liste de chaîne de caractères contenant toutes les informations nécessaires pour un organisme. Nous avons choisi de procédé ainsi afin de faciliter la récupération des résultats des requêtes SQL: les résultat sont d'abord entrés dans une variable de type « String [] », puis grâce au constructeur que nous avons défini, il est très facile de passer de cette variable à une autre variable du type voulu, « Organism » dans notre exemple. Certaines classes comme « Molecule » ont quelques attributs de type « Hashtable » et donc dans le constructeur, ces variables de type « Hashtable » sont juste déclarées, elles seront remplies d'informations plus tard grâce à plusieurs autres méthode. Effectivement, certaine de ces classes possèdent des méthodes de une ou deux lignes, du type:

```
public void setComp(Compartment c) {
       comp = c;
}
```

Ce sont des méthodes très simples qui sont principalement nécessaires pour récupérer les attributs privés des classes, ou pour pouvoir les initialisés. Toutes ces méthodes sont appelées principalement par la classe « Network ».

Mise en Fonction

Etat d'avancement du projet

A l'heure de l'écriture de ce rapport, le logiciel n'est pas encore terminé. Tous les éléments abordés dans ce rapport sont bien évidemment déjà intégrés, mais certaines retouches ont déjà du être effectuées au niveau des fenêtres et au niveau de certaines classes, suite a de nouvelle demande de Julien Berro.

La sauvegarde de fichier au format SBML n'est pas encore implémentée. Certaines méthodes sont déjà écrites, mais non testées, ni intégrées au logiciel. Le stage ayant commencé une semaine plus tard que la date demandée, il finira également une semaine plus tard. Ainsi, nous devrons avoir le temps de compléter le logiciel afin qu'il soit utilisable, même si toutes les fonctionnalités ne sont pas intégrées. A priori, la sauvegarde de fichiers au format SBML devrait pouvoir être implémentée, tout comme la sauvegarde du fichier de suivi.

Tests effectués et Manuel d'utilisateur

Pendant toute la durée de la réalisation du logiciel, de nombreux tests ont été effectués à chaque étape du développement. L'objectif visé était d'essayer de toujours avoir une version fonctionnelle du logiciel. Pour toutes les étapes effectuées jusqu'à présent, la rédaction d'un jeu de tests précis ne s'est pas montrée nécessaire, vu que le logiciel ne gère que des consultations de la base de données. Avec le client de base de données PostgreSQL « psql », il suffit de comparer ce que contiennent les tables de la base de données, avec ce que le logiciel nous affiche pour vérifier si les différentes requêtes SQL qui permettent les consultations de la base de données soient exactes.

Ces tests nous permettent également de tester que tous les choix fais par l'utilisateur sont bien pris en compte. Typiquement, les compartiments qui apparaissent dans la fenêtre et pouvant être sélectionnés doivent uniquement être ceux présents dans la location choisie juste précédemment. De même, les réactions qui s'affichent dans la fenêtre principale doivent être uniquement celles sélectionnées précédemment.
Une attention particulière a donc été portée sur ces tests, afin de toujours garder une cohérence durant le développement du logiciel.

Lors de l'implémentation de sauvegarde du fichier au format SBML, et du fichier de suivi, il sera sûrement nécessaire de rédigé un bon jeu de test car plusieurs conditions seront a prendre en compte, et donc, il faudra être précis dans les tests.

Un manuel d'utilisateur a également été effectué. Vous pouvez le consulter en annexe (Annexe 4, page 50), ce n'est pas une version définitive étant donnée que le logiciel n'est pas terminé, il sera donc nécessaire de faire quelques modifications.

Problèmes rencontrés

De multiples problèmes ont été rencontrés durant le stage. Le problème principal rencontré pendant ce stage, fut de travailler en parallèle avec les travaux effectués sur la base de données PathwayDB. Effectivement, au début du stage, la base de données n'était pas définitive, et donc elle a changé tout au long du stage, puisque c'était justement lors de différentes questions et réunions avec Julien Berro que l'on s'est aperçu qu'il y avait des tables à rajouter, des champs à changer ou à supprimer.

Même si effectuer ces changements dans la base n'était pas de notre ressort, cela nous a obligé à revoir plusieurs éléments de notre logiciel, comme les requêtes SQL ou des attributs de certaines classes, voir en créer d'autres. Afin de pouvoir plus facilement appréhender ces changements soudains, plusieurs bases de données de test sur le modèle de PathwayDB furent créées. Ainsi, lorsqu'un changement devait être effectué sur la base, nous avions la possibilité de finir ce qui était en train d'être implémenté, pour après s'occuper uniquement de rendre le logiciel compatible avec les changements effectués pendant ce temps sur PathwayDB. Ensuite, une fois le logiciel de nouveau totalement fonctionnel sur PathwayDB, les autres bases de données de test étaient à leur tour mises à jour avec les nouveaux changements.

Un autre problème majeur s'est présenté aussi un peu plus tard dans le stage. Pour pouvoir implémenter la sauvegarde du réseau créé par le logiciel dans un fichier au format SBML, la librairie SBML pour Java est nécessaire. Cependant, celle-ci, au début du stage, n'existait pas encore et aurait dû sortir vers la fin du mois d'Avril. Malheureusement, celle-ci n'est finalement pas sortie à la date annoncé, et donc, nous avons décidé de l'attendre encore un peu, en espérant une sorti imminente. Vers le 15 Mai, la librairie n'étant toujours pas sorti, nous avons donc décidé de chercher une alternative. Un logiciel utilisant le format SBML, en Java et en « Open Source » a été trouvé, l'auteur américain a écrit lui même ses classes pour générer et lire les fichiers SBML. Après discussion avec Julien Berro, et la réponse positive de l'auteur à un e-mail lui demandant l'autorisation d'utiliser ses sources, nous avons donc décidé d'utiliser le code de ce logiciel pour permettre à notre logiciel de pouvoir générer des fichiers au format SMBL.

Modifications effectuées et prise en compte des remarques des utilisateurs.

Ce logiciel a été réalisé dans l'idée d'essayer de coller au maximum aux désirs de Julien Berro et de Jean-Louis Martiel, dans la mesure du possible. Leurs remarques ont donc été prises en compte très sérieusement, donc de très nombreuses modifications ont été apportées au logiciel en cours de réalisation en fonctions de ces remarques. En effet, leurs volontés étaient, au départ, assez floue. Donc en voyant l'évolution du logiciel, leurs idées devenaient de plus en plus claire, et donc de multiples remarques ou changement d'avis sont apparus concernant l'interface, comme par exemple l'affichage des lois cinétiques par leur nom ou leur expression symbolique, ou les différentes sélections possibles, comme par exemple pouvoir sélectionner les réactions voulues toutes les prendre en compte à partir du choix des molécules effectué.
De multiples changements de ce type ont donc du intervenir durant le développement du logiciel, entraînant des modification plus ou moins conséquentes dans le logiciel, retardant ainsi dans l'avancement du projet.

Conclusion

Le but de ce projet était de créer un logiciel permettant de créer un réseau de régulation biochimique générant des fichiers au format SBML à partir d'une base de données. Au début, la base de données en PostgreSQL existait déjà mais elle était non définitive. Il a fallu donc créer un logiciel en Java autour de cette base de données. Cela est possible grâce à la librairie JDBC permettant un dialogue entre un la base de données est le logiciel en Java. Une conception complète en UML a été réalisée afin de posséder une vue d'ensemble et avoir une idée de la façon de procéder pour réaliser ce logiciel. Grâce à cette conception, deux grandes parties sont apparues clairement: celle du noyau applicatif, et celle de l'interface Homme-Machine.

Etant donné le projet que nous avons réalisé en binôme, ces deux grandes parties ont permis une répartition du travail efficace et logique. C'est donc ainsi que le travail fut réalisé, ces deux parties étant réalisé en parallèle, un gain de temps considérable a été obtenu grâce à cette répartition. La partie du noyau applicatif m'ayant été attribué, ce rapport a abordé uniquement cette partie, expliquant ainsi le fonctionnement de la classe Network principalement, et le continuel dialogue entre une base de données PostgreSQL avec un logiciel en Java. Le logiciel a été développé en parallèle avec la base de données, donc, chaque modification de cette dernière entraînait des modifications à effectuer afin de garder le logiciel fonctionnel.

La librairie libSBML pour Java, nécessaire pour générer un fichier SBML s'est fait attendre tout au long du stage, et elle n'est, à l'heure actuelle toujours pas disponible. Pour parer ce problème, le code source d'un autre logiciel en Java générant des fichiers SBML sera utilisé pour notre logiciel. Grâce à une coopération étroite avec Julien Berro, le logiciel a toujours pu avancer dans l'idée souhaitée du projet. Des tests ont également été réalisés à chaque étape, et un manuel d'utilisateur non définitif a également été rédigé. Cependant, le logiciel est, à l'heure actuelle pas encore terminée, la sauvegarde au format SBML et de la sauvegarde du fichier de suivi n'ont pas encore été implémenté, cependant, étant donnée la fin tardive du stage, cela devrait être réalisé dans les temps.

Lors de ce stage, évidemment quelques difficultés ont été rencontrées surtout au début du stage. Premièrement, n'étant pas habitué à la configuration et installation de logiciel dans un environnement Linux, un petit temps d'adaptation fut nécessaire pour le premier jour. Il a fallu aussi un temps d'adaptation pour l'utilisation de la librairie JDBC. D'ailleurs quelques recherches ont du être effectuées sur Internet afin de trouver des tutoriaux clairs est précis sur l'utilisation de cette librairie.

Les connaissances acquissent à l'IUT ont été d'une grande utilité pour plusieurs domaines durant tout le déroulement du stage. Ces connaissances nous ont permis de comprendre beaucoup plus rapidement l'organisation de la base de données PathwayDB, la logique utilisée dans JDBC, et les requêtes SQL ont ainsi été écrite beaucoup plus rapidement, et correctement. Ainsi, aucun problème au niveau base de données n'a été rencontré. Les connaissances acquissent en Java et en UML ont été également d'une grande utilité, ainsi que l'utilisation de Jbuilder à l'IUT, cela nous a permis de ne pas perdre de temps pour découvrir le logiciel de développement. Ce stage m'a permis principalement de renforcer mes connaissances en Java et en base de données. Il m'a permis également de cerner le travail sur un projet gravitant autour d'un autre projet lui même non terminé. Mais, au final, le stage ne s'est pas déroulé de manière tellement différente que les projets effectués au sein de l'IUT, et donc le passage de la théorie à la pratique professionnel s'est déroulé de manière naturelle.

Glossaire

Compartiment : Les locations peuvent encore être décrite génétiquement par des compartiments, qui sont de taille précise et donc permettent de travailler avec un nombre ou une concentration d'éléments.

Location : Différentes parties qu'on nommera locations découpent un organisme. Par exemple dans l'organisme « Homme », on trouvera la location « neurone ».

Loi cinétique : Chaque réaction dispose de différentes lois cinétiques, qui régissent la manière dont les molécules interagissent entre elles lorsque la réaction se produit. Il y a plusieurs lois cinétiques possibles pour une réaction.

Molécule : Chaque molécule dispose d'un nom, et peut avoir plusieurs « alias », ou molécules similaires mais de nom différent, qui se rapportent à la même molécule. Chaque molécule peut elle avoir plusieurs concentration initiale, selon la location et le compartiment dont elle est issue.

Réaction : Chaque réaction dispose d'une expression symbolique, et des réactifs et produits (les différentes molécules qui interviennent dans la réaction). Une réaction peut intervenir entre plusieurs molécules de locations ou compartiments différents.

Réseau de régulation biochimique : C'est un graphe permettant de représenter les interactions chimiques entre différentes molécules dans une partie d'un organisme vivant.

Serveur : Il s'agit d'une machine physique accessible à partir d'autres machines.

SGBD : Initiales de Système de Gestion de Bases de Données. Cela permet de dialoguer avec la base de données.

Taxon : Chaque organisme peut appartenir à une « famille » d'organismes, nommé taxon, par exemple les mammifères.

Bibliographie

- Borland Jbuilder : Version personnelle, gratuite en téléchargement sur leur site officiel « http://www.borland.fr/ », ainsi que la documentation associée, très utile.

- Sun JDBC : Site officiel de Sun sur la bibliothèque JDBC de gestion de bases de données : « http://java.sun.com/products/jdbc/ ». On y trouve toutes les références ainsi que la documentation et des exemples sur l'utilisation de cette bibliothèque.

- PostgreSQL : Site sur le SGBD PostgreSQL, « http://www.postgresql.org » propose de nombreuses documentations, ainsi qu'une partie consacrée à JDBC « http://jdbc.postgresql.org » et plus précisément au « Driver » spécifique à PostgreSQL pour JDBC.

- « Le tout en poche - SQL » : Livre aux éditions CampusPresse par Ronald R. Plew et Ryan K. Stephens sur le langage SQL, principalement utilisé pour créer les requêtes nécessaires.

- « Programmer en Java » : Livre aux éditions Eyrolles par Claude Delannoy, présentant la programmation en Java et Swing.

- JDesigner : Site Internet du logiciel JDesigner, « http://www.cds.caltech.edu/~hsauro/JDesigner.htm » sur lequel on peut télécharger le logiciel JDesigner, sa documentation, et les sources du logiciel.

- SBML : site Internet de la bibliothèque SBML : « http://sbml.org/ » sur lequel on peut télécharger la librairie pour C++, et aussi la documentation complète sur le format SBML.

ANNEXES

Annexe 1 : Le planning de travail

	Planning prévisionnel	Planning réalisé
Semaine 1	Installation et configuration des machines de développement.	Installation et configuration des machines de développement.
	Installation et configuration d'un client PostgreSQL, un logiciel CVS, de Jbuilder et des différents libraires nécessaires.	
Semaine 2	Accès à une base de données distante	Conception
	Une conception a été faite en UML au lieu de commencer directement la réalisation, car au vu de la base de données, il semblait évident que le logiciel allait posséder un certain nombre de classe. Cette conception a déplacé toutes les prévisions d'une semaine.	
Semaine 3	Aide au remplissage des fenêtres des molécules.	Connexion à la base de données
	La réalisation du logiciel a commencé par la connexion à la base de données à l'aide de JBDC.	
Semaine 4	Aide au remplissage de la fenêtre des propriétés des réactions	Choix dans la liste des molécules.
	Toutes les étapes allant du choix du mode de sélection au choix des molécules ont été réalisées. Ceci prend donc en compte le choix des organismes ou taxons, des locations et des compartiments.	
Semaine 5	Aide au remplissage de la fenêtre des propriétés des compartiments	Choix de la liste des réactions.
	Implémentation de toutes les méthodes utiles pour la sélection des réactions, ainsi que de l'interface. Affichage des réactions dans la fenêtre a aussi été implémenté.	
Semaine 6	Stockage des informations entrées dans les différentes fenêtres de propriétés.	Rajout d'élément.
	Implémentation du bouton « continue » permettant donc de pouvoir rajouter une location et un compartiment dans le réseau ainsi que ses molécules.	
Semaine 7	Création d'un réseau	Finalisation du réseau. Rédaction du manuel d'utilisateur et du début du rapport.
	Implémentation de la fenêtre permettant d'éditer et de vérifier les informations du réseau, ainsi que de l'interface pour la sauvegarde du fichier SBML. Rédaction de l'introduction et de a première partie de ce rapport.	
Semaine 8	Ajout de fonctionnalité de lecture d'un fichier SBML	Rédaction du rapport de stage
	Rédaction et impression entière du rapport de stage et des annexes	
Semaine 9	Mise en place d'une procédure simple d'installation sur différent support.	[Prévision] Implémentation de la sauvegarde des fichiers en SBML et du fichier de suivi. Présoutenance de stage.
	Implémentation des fichiers de sauvegarde au format SBML et du fichier de suivi. Présoutenance de stage avec J.Berro et J.L.Martiel.	
Semaine 10	Finalisation et correction de divers problèmes rencontrés.	[Prévision] Mise en place d'une procédure simple d'installation et de configuration du logiciel.

Annexe 2 : La méthode setMolecules

```
public void setMolecules(Hashtable list) throws Exception {
    String am;
    int j;
    Hashtable selectedMolecules = list;
    log.write("Molecules selected :");
    Enumeration enum3;
    Enumeration enum = selectedMolecules.elements();
    while (enum.hasMoreElements()) {
      Molecule mol = (Molecule) enum.nextElement();
      log.write(" - " + mol.id + " | " +
                mol.name + " | " + mol.type +
                " | " + mol.info);
    }
    log.write("Molecules values :");
    ResultSet rs;

    try {
      enum = selectedMolecules.elements();
      while (enum.hasMoreElements()) {
        Molecule mol = (Molecule) enum.nextElement();
        j = 0;
           String query = "SELECT DISTINCT ic.id, ic.initial_value, un.name,
           ic.boundarycondition, ic.constant, un.id FROM has_ini_cond hic,
           initial_conditions ic, location loc, molecule mol, unit un WHERE
           hic.id_ini_cond = ic.id AND hic.id_mol = mol.id AND hic.id_location =
           loc.id AND un.id = ic.id_unit AND loc.id IN ( ";
        query += activeLocation.id + " ) AND mol.id in ( ";
        query += mol.id;
        if (mol.hasAliases()) {
          Enumeration enum2 = mol.getAliasesElements();
          while (enum2.hasMoreElements()) {
            query += " , " + ( (Molecule) enum2.nextElement()).id + " ";
          }
        }
        query += " ) ";
        if (mol.numProperties > 0) {
          query += " AND ic.id NOT IN ( ";
          j = 0;
          for (int i = 0; i < mol.numProperties; i++) {
            if (i > 0) {
              query += " , ";
            }
            query += "'" + mol.getPropID(i) + "'";
          }
          query += " ) ";
        }
        rs = stmt.executeQuery(query);
        j = 0;
        while (rs.next()) {
          j++;
        }
        if (j == 0) {
          comp.addMolecule(mol);
          continue;
        }
        rs.first();
        j = 0;
```

```
        do {
          String r = rs.getString(3);
          if (Character.getNumericValue(r.charAt(0)) < 0) {
            r = "m" + r.subSequence(1, r.length());
          }
          mol.addProperty(rs.getString(1), Float.parseFloat(rs.getString(2)),
                          r,
                          rs.getString(4).equals("f") &&
                          rs.getString(5).equals("f"), rs.getString(6), comp);
          log.write("Molecule " + mol.name + " : Concentration = " +
                    mol.getConc(j) + " " + mol.getUnit(j));
          j++;
        }
        while (rs.next());
        comp.addMolecule(mol);
      }
    }
    catch (Exception e) {
      e.printStackTrace();
      throw new Exception(e.getMessage());
    }
  }
```

Annexe 3 : La méthode getReactionList

```
public Hashtable getReactionsList() throws Exception {
    String[][] reacs;
    String query = "";
    Reaction[] reactions;
    ResultSet rs;
    Hashtable ht = new Hashtable();
    int i = 0;
    int j;
    int o = 0;
    Compartment cp;

    try {
      Enumeration enum, enum2;
      query =
       "SELECT * FROM reaction WHERE id IN ( ( ( ( SELECT id_reac FROM reacts ";
      query += " WHERE id_mol IN ( ";
      Enumeration enum1 = selectedCompartments.elements();
      i = 0;
      while (enum1.hasMoreElements()) {
        cp = (Compartment) enum1.nextElement();
        enum = cp.getMoleculesElements();
        while (enum.hasMoreElements()) {
          Molecule mol = (Molecule) enum.nextElement();
          if (i > 0) {
            query += " , ";
          }
          query += " '" + mol.id + "' ";
          enum2 = mol.getAliasesElements();
          while (enum2.hasMoreElements()) {
            query += ", '" + ( (Molecule) enum2.nextElement()).id + "' ";
          }
          i++;
        }
      }
      enum1 = selectedCompartments.elements();
      i = 0;
      query += " ) ) INTERSECT ( SELECT id_reac FROM products ";
      query += " WHERE id_mol IN ( ";
      while (enum1.hasMoreElements()) {
        cp = (Compartment) enum1.nextElement();
        enum = cp.getMoleculesElements();
        while (enum.hasMoreElements()) {
          Molecule mol = (Molecule) enum.nextElement();
          if (i > 0) {
            query += " , ";
          }
          query += " '" + mol.id + "' ";
          enum2 = mol.getAliasesElements();
          while (enum2.hasMoreElements()) {
            query += ", '" + ( (Molecule) enum2.nextElement()).id + "' ";
          }
          i++;
        }
      }
      query += " ) ) ) EXCEPT ( ( SELECT id_reac FROM reacts ";
      query += " WHERE id_mol NOT IN ( ";
      enum1 = selectedCompartments.elements();
      i = 0;
      while (enum1.hasMoreElements()) {
        cp = (Compartment) enum1.nextElement();
        enum = cp.getMoleculesElements();
```

```
  while (enum.hasMoreElements()) {
    Molecule mol = (Molecule) enum.nextElement();
    if (i > 0) {
      query += " , ";
    }
    query += " '" + mol.id + "' ";
    enum2 = mol.getAliasesElements();
    while (enum2.hasMoreElements()) {
      query += ", '" + ( (Molecule) enum2.nextElement()).id + "' ";
    }
    i++;
  }
}
query += " ) ) UNION ( SELECT id_reac FROM products ";
query += " WHERE id_mol NOT IN ( ";
enum1 = selectedCompartments.elements();
i = 0;
while (enum1.hasMoreElements()) {
  cp = (Compartment) enum1.nextElement();
  enum = cp.getMoleculesElements();
  while (enum.hasMoreElements()) {
    Molecule mol = (Molecule) enum.nextElement();
    if (i > 0) {
      query += " , ";
    }
    query += " '" + mol.id + "' ";
    enum2 = mol.getAliasesElements();
    while (enum2.hasMoreElements()) {
      query += ", '" + ( (Molecule) enum2.nextElement()).id + "' ";
    }
    i++;
  }
}
query += " ) ) ) ) ";
query += " ) ";
System.out.println(query);
rs = stmt.executeQuery(query);
if (rs == null) {
  return null;
}
i = 0;
while (rs.next()) {
  i++;
}
if (i == 0) {
  return null;
}
reacs = new String[i][numReactionTitles];
reactions = new Reaction[i];
rs.first();
j = 0;
do {
  for (int k = 0; k < numReactionTitles; k++) {
    reacs[j][k] = rs.getString(k + 1);
  }
  reactions[j] = new Reaction(reacs[j]);
  j++;
}
```

```
      while (rs.next());
      for (i = 0; i < reactions.length; i++) {
        query = "SELECT DISTINCT molecule.name, reacts.stoechiometry FROM
reacts, reaction, molecule WHERE reacts.id_reac = reaction.id and reacts.id_mol
= molecule.id and reaction.id = '" + reactions[i].id + "'";
        log.write("Getting reactifs stoechiometry for reaction " +
                  reactions[i].name + "...");
        rs = stmt.executeQuery(query);
        j = 0;
        while (rs.next()) {
          j++;
        }
        rs.first();
        j = 0;
        do {
          reactions[i].addReac(rs.getString(1), rs.getString(2));
          j++;
        }
        while (rs.next());

        if (reactions[i].symbExpr.lastIndexOf("=") == -1) {
          reactions[i].type = false;
        }
        else {
          reactions[i].type = true;
        }
        rs.close();
        query = "SELECT DISTINCT molecule.name, products.stoechiometry FROM
products, reaction, molecule WHERE products.id_reac = reaction.id AND
products.id_mol = molecule.id AND reaction.id = '" + reactions[i].id + "'";
        rs = stmt.executeQuery(query);
        j = 0;
        while (rs.next()) {
          j++;
        }
        rs.first();
        do {
          reactions[i].addProd(rs.getString(1), rs.getString(2));
        }
        while (rs.next());
      }
      for (i = 0; i < reactions.length; i++) {
        setKineticLawList(reactions[i]);
      }
      Hashtable result = new Hashtable();
      for (i = 0; i < reactions.length; i++) {
        result.put(reactions[i].id, reactions[i]);
      }
      Hashtable list = result;
      selectedReactions = new Hashtable();
      enum = list.elements();
      int num = 0;
      while (enum.hasMoreElements()) {
        Reaction reac = (Reaction) enum.nextElement();
        Hashtable temp = nextReac(reac, 0);
        enum2 = temp.elements();
        while (enum2.hasMoreElements()) {
          Reaction re = (Reaction) enum2.nextElement();
          Enumeration enum3 = reac.getLawElements();
          while (enum3.hasMoreElements()) {
            re.addLaw( (KineticLaw) enum3.nextElement());
          }
          formatReaction(re);
          selectedReactions.put(new Integer(num), re);
```

```
          num++;
        }
      }

      return selectedReactions;
    }
    catch (Exception e) {
      throw new Exception(e.getMessage());
    }
  }
```

Annexe 4 : Manuel d'utilisateur (non définitif)

Ce logiciel vous permet de générer un fichier au format SBML d'un réseau de régulation biochimique que vous aurez au préalable spécifié. Vous pouvez créer des réseaux avec plusieurs locations et compartiments mettant en jeu de nombreuses molécules et des réactions inter-compartiment ou non.
Pour cela, le logiciel se découpe en plusieurs étapes. Ce manuel va vous permettre de vous guider durant les différentes étapes.

1ère étape : la connexion

Lors de l'exécution du logiciel, vous allez voir apparaître la fenêtre de connexion à la base de données de ce type.

Il y a 4 champs à remplir:
- Server Name: il faut mettre ici le nom du serveur où se trouve la base de données.
- Database Name : il faut mettre ici le nom de la base de données à laquelle vous voulez vous connecter.
- Login Name : il faut mettre votre nom d'utilisateur de la base de données.
- Password : il faut rentrer votre mot de passe de connexion à la base de données.

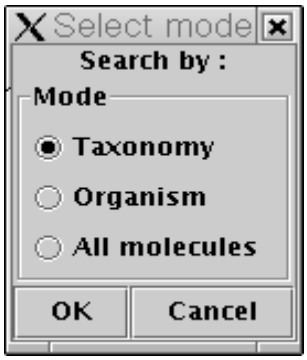
Le nom du serveur (server name) et le nom de la base de données (database name) se trouvent également dans un fichier « serveur.ini » ce qui évite d'avoir à ré écrire à chaque démarrage du logiciel. Pour éditer ce fichier ini, utiliser un logiciel de traitement de texte standard; la première ligne correspond au nom du serveur, la deuxième correspond au nom de la base de données.

Une fois tous les champs remplis, cliquez sur le bouton « Connect » pour vous connecter à la base de données. Si tous les champs sont correctement remplis, la fenêtre suivante va s'ouvrir. Vous pouvez cliquer sur le bouton « Cancel » si vous souhaiter annuler l'opération.

2éme étape : le choix du domaine

Après vous être connecté à la base de données une fenêtre vous prévenant que toutes les données rentrées et non sauvegardées auparavant seront effacées. Lors de la première utilisation du logiciel après son exécution, choisissez « yes » pour continuer.

Ensuite la fenêtre du choix du domaine de travail va s'ouvrir. Cette fenêtre se présente ainsi:



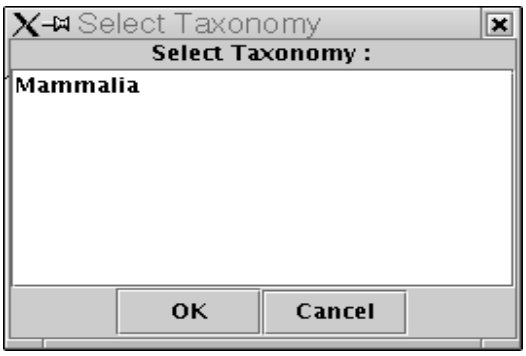
Vous pouvez sélectionner un des trois choix qui s'offre a vous:

- Par taxon (Taxonomy): vous pourrez ensuite choisir le taxon sur lequel vous souhaitez travailler.
- Par organisme (Organism): Tout les choix suivants seront par rapport à un organisme que vous choisirez après.
- Ou alors, vous pouvez choisir de ne pas spécifier de domaine de travail, et donc d'avoir le choix, plus tard, avec tous les paramètres de toutes les molécules possibles (All molecules), si vous effectuez ce choix, passez directement à l'étape 6.

Une fois votre choix effectué, appuyez sur le bouton « OK » pour continuer la création du réseau, ou vous pouvez cliquer sur « Cancel » pour annuler la création de ce réseau.

3ème étape: choix de l'organisme ou du taxon

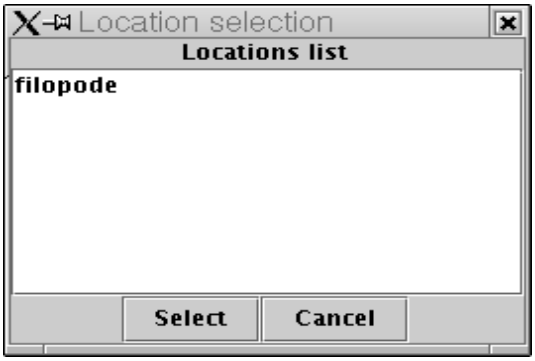
Après avoir fait votre choix du domaine de travail, vous allez pouvoir choisir le taxon ou l'organisme dans lequel vous comptez travailler. Dans les deux cas, une fenêtre de ce type va s'ouvrir:



Vous pouvez donc choisir un organisme ou taxon parmi tous ceux disponible dans la base de données. Une fois votre choix effectué, cliquez sur le bouton « OK » afin de valider.
Si vous voulez tout annuler, vous pouvez en cliquant sur le bouton « Cancel ».

4ème étape: choix de la location

Vous devez ensuite sélectionner une location. Une fenêtre de ce type s'ouvre:



Toutes les locations disponibles dans la base de données par rapport à l'organisme ou au taxon sélectionné sont disponibles. Vous pouvez donc choisir une location parmi toutes celles disponibles dans la liste. Appuyez sur le bouton « Select » pour valider.
Vous pouvez tout annuler en cliquant sur le bouton « Cancel »

5ème étape: choix du compartiment

Après avoir choisi une location, vous pouvez choisir parmi les différents compartiments disponibles pour cette location. Une nouvelle fenêtre de ce type s'ouvre.

Compartment selection
Compartments list
autre filopode (Size : 2 ; Info : ca)
exemple de filopode (Size : 1 ; Info : n'importe quoi)
Select
Cancel

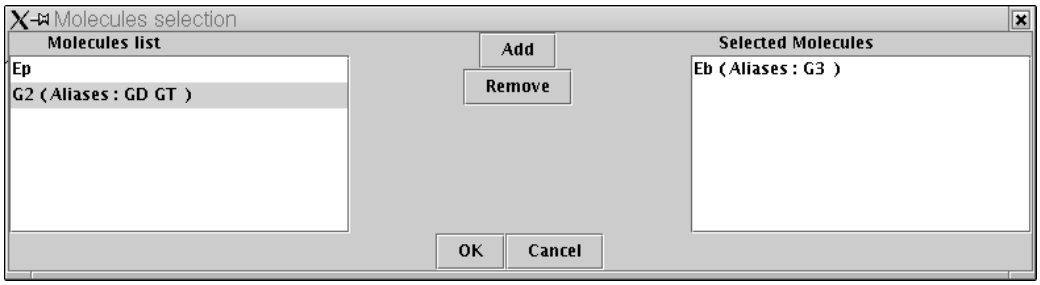
Vous devez donc choisir un compartiment dans lequel toutes les molécules que vous choisirez plus tard seront contenues. Cette fenêtre est redimensionnable afin que les informations sur les compartiments soient complètement lisibles. Une fois votre choix effectué, cliquez sur le bouton « Select » pour continuer, ou sur « Cancel » pour tout annuler.

6ème étape: choix des molécules

Ensuite, une nouvelle fenêtre s'ouvre, c'est celle qui va vous permettre de sélectionner les molécules qui seront soit:

- Si vous avez choisi de travailler dans un organisme ou un taxon précis, le choix que vous allez effectuer sera les molécules se trouvant dans le compartiment choisi auparavant.
- Si vous avez choisi de ne pas spécifier de domaine de travail, alors le choix que vous allez effectuer sera toutes les molécules mises en jeu dans le réseau de régulation que vous créez.

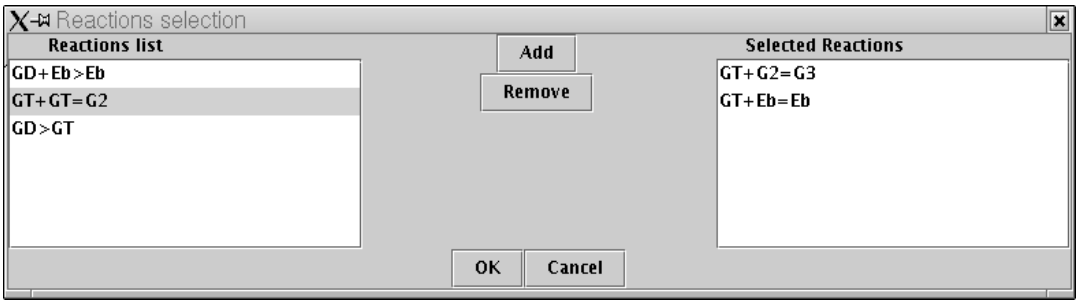
La fenêtre du choix des molécules se présente ainsi.



Dans la liste de gauche se trouvent toutes les molécules disponibles dans la base de données. Entre parenthèses sont indiqué les différents alias que la molécule peut avoir, c'est à dire, différents noms. Dans la liste de droite se trouvent toutes les molécules que vous avez sélectionnées. Pour sélectionner une molécule, il suffit de la sélectionner dans la liste de gauche est de cliquer sur le bouton « Add » (ajouter) pour faire passer la molécule dans la liste de droite « Selected Molecules ».
Inversement, si vous voulez enlever une molécule que vous avez choisie, sélectionnez la dans la liste de droite et cliquez sur le bouton « Remove ». Une fois que vous avez sélectionné toutes les molécules que vous vouliez, cliquez sur le bouton « OK » pour continuer. Si vous appuyez sur « Cancel », toute la création du réseau sera annulée et perdue.

7ème étape: choix des réactions

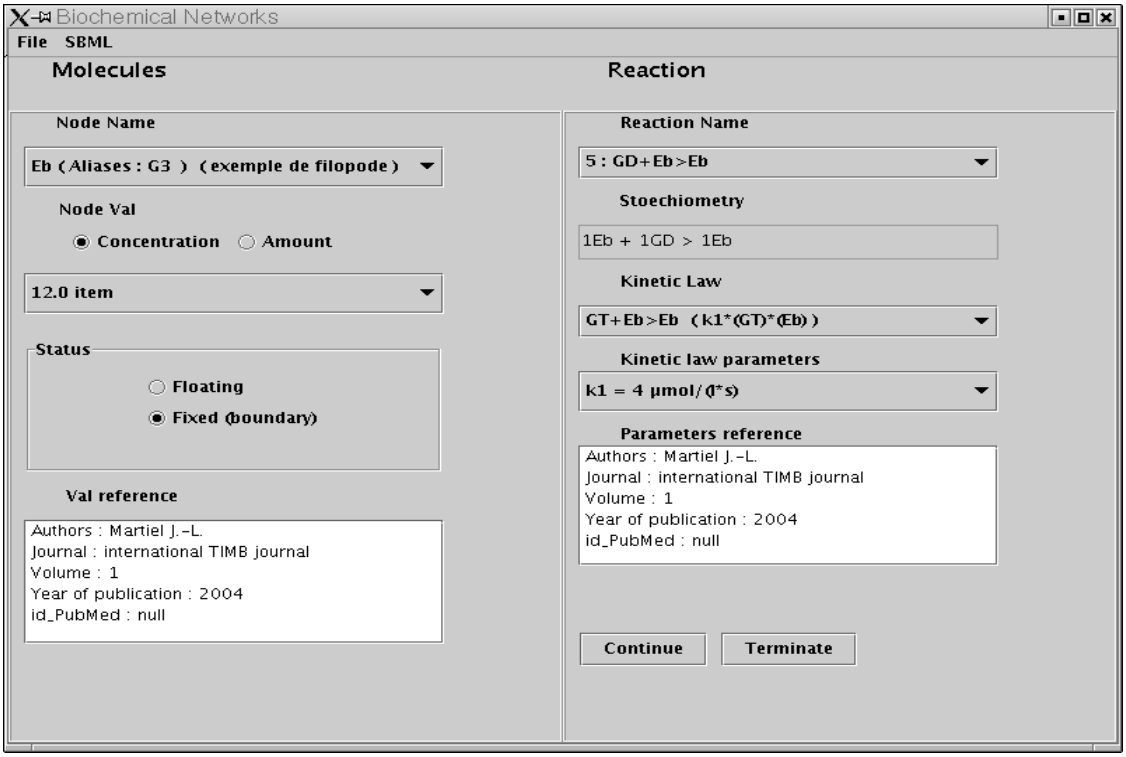
Après avoir choisi les molécules, une fenêtre, vous permettant de choisir les réactions que vous désirez voir dans votre réseau de régulation biochimique, va s'ouvrir. Cette fenêtre est de ce type:



Dans la liste de gauche se trouvent toutes les réactions disponibles dans la base de données en fonction des molécules sélectionnées auparavant, et même avec celles d'autres compartiments si vous en aviez déjà choisi auparavant. Dans la liste de droite, se trouve la liste de toutes les réactions sélectionnées. Cette fenêtre a le même fonctionnement que la fenêtre du choix des molécules.
Pour sélectionner une réaction, mettez la en sur brillance dans la liste de gauche et cliquez sur le bouton « Add ». Pour dé sélectionner une réaction, mettez la en sur brillance dans la liste de droite, et cliquez sur « Remove ».
Une fois votre choix effectué, cliquez sur « OK » pour continuer, ou sur « Cancel » pour tout annuler.

8ème étape: choix des paramètres des molécules et des lois cinétiques

Après avoir effectué tous ces choix, la fenêtre principale du programme s'affiche. Cette fenêtre vous permet d'avoir une vue d'ensemble sur le réseau que vous avez créé, et vous permet de sélectionner les derniers éléments, c'est à dire les paramètres des molécules et les lois cinétiques associés aux réactions voulues, ainsi que leur paramètres. Cette fenêtre se présente ainsi.



De multiples informations sont affichées dans cette fenêtre :

A gauche se trouvent les informations sur les molécules.
La première liste déroulante en haut à gauche de la fenêtre contient toutes les molécules que vous avez sélectionnées. Entre parenthèses s'affiche le compartiment dans lequel se trouve la molécule. Vous pouvez en sélectionner une en particulier.
Les différentes valeurs possibles pour cette molécule se trouvent dans la seconde liste déroulante et vous pouvez sélectionner une de ces valeurs pour la donner à la molécule dans votre réseau de régulation.
Pour les valeurs, vous pouvez sélectionner soit la concentration, soit la quantité (Amount).
Si vous avez choisi un domaine de travail, les valeurs disponibles seront uniquement celles disponibles dans la location où se trouve la molécule, et la quantité (Amount) est la concentration divisée par la taille du compartiment.
Si vous n'avez pas choisi de domaine de travail, les valeurs disponibles sont toutes les valeurs possibles pour une molécule quelle que soit la location, et il n'y a pas de quantité (Amount), vu qu'aucun compartiment n'a été sélectionné.
Le statut (status) est également défini par le choix de la valeur, et vous pouvez également le changer.
La dernière information concernant les molécules, est la référence de la valeur. Ici sont affichées les références sur la provenance de la valeur de la molécule. Vous ne pouvez pas les modifier.

A droite se trouvent les informations sur les réactions.
La première liste déroulante se trouve en haut à droite de la fenêtre. Elle contient toutes les réactions que vous avez sélectionnées, et vous pouvez en sélectionner une. Son expression complète avec sa stoechiométrie exacte s'affichera juste en dessous.
Ensuite, vous pouvez sélectionner une loi cinétique dans la réaction choisie grâce à la liste déroulante « Kinetic Law ».
Une fois la loi cinétique choisie, vous pourrez choisir un de ses paramètres parmi tous ceux disponibles dans la base de données.
La dernière information qui concerne les réactions est l'affichage de la référence du paramètre de la loi cinétique actuellement sélectionnée.

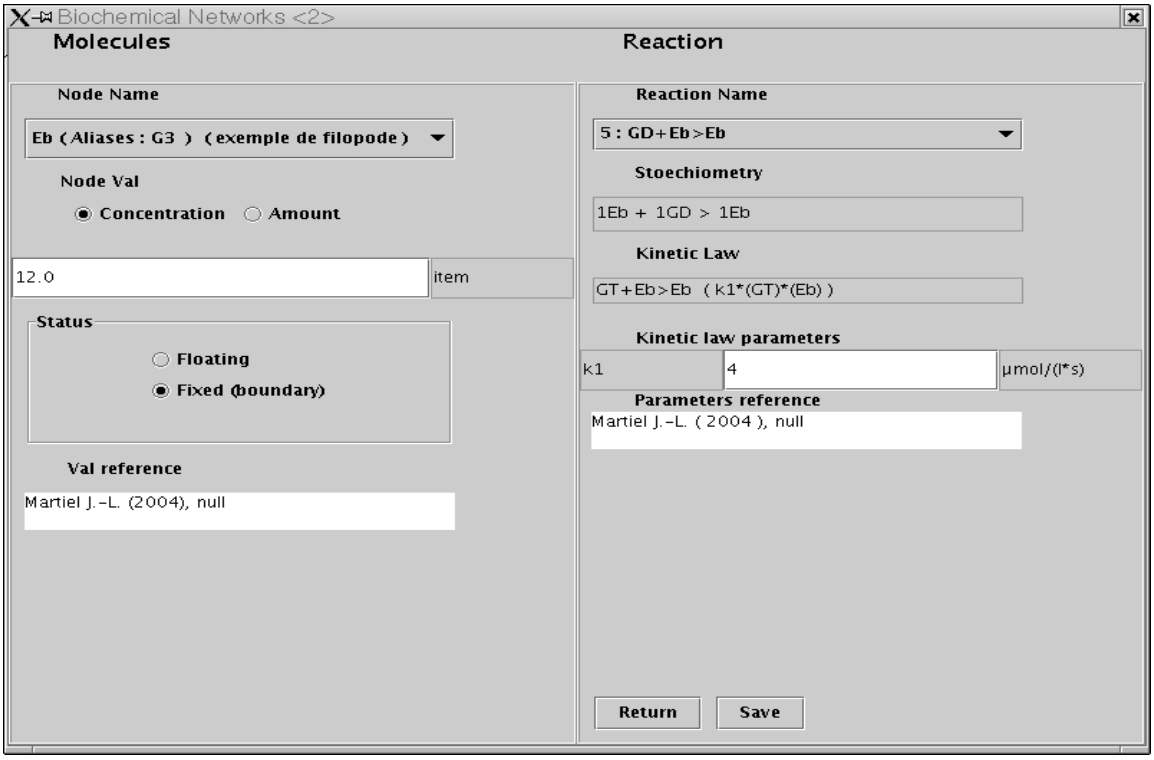
Si vous avez choisi de travailler dans un domaine spécifique, organisme ou taxon, vous aurez le choix de continuer en appuyant sur le bouton « Continue », ou, si vous désirez vous arrêter là, ou alors vous avez choisi de ne pas spécifier de domaine de travail, vous pouvez cliquez sur « Terminate » pour passer dans l'étape de finition du réseau.

9ème étape: rajout de location et compartiment dans le réseau

Si vous avez choisi de continuer, vous allez de nouveau vous retrouver sur la fenêtre du choix des locations, vous pouvez donc choisir de nouveau la même location ou une autre (voir étape 4).
Ensuite vous pourrez de nouveau sélectionner un compartiment, sauf ceux que vous aviez déjà sélectionnés auparavant si vous avez sélectionné la même location (voir étape 5).
Puis vous allez de nouveau sélectionner les molécules qui se trouveront dans ce compartiment (voir étape 6).
Puis la fenêtre du choix des réactions va de nouveau s'ouvrir, avec toutes les réactions disponibles dans la base par rapport à toutes les molécules que vous avez choisies depuis le début de la création de votre réseau. Vous devez également sélectionner de nouveau les réactions que vous aviez déjà sélectionnées auparavant.
Ensuite vous allez de nouveau avoir la fenêtre principale avec toutes les nouvelles informations que vous avez rentrées, et vous pouvez, si vous le désirez, encore rajouter un nouveau compartiment ou une nouvelle location à votre réseau en cliquant sur « Continue », ou alors passer à l'étape de finition en cliquant sur « Terminate ».

10ème étape: finition du réseau.

Une fois tous vos choix effectués, vous allez alors voir la fenêtre de finition s'ouvrir. Cette fenêtre est de ce type:



Cette fenêtre ressemble beaucoup a la précédente, elle comporte d'ailleurs les mêmes informations. Sauf qu'ici, vous ne pouvez plus sélectionner la valeur de la molécule sélectionnée, mais vous pouvez directement éditer cette valeur.
Il en est de même pour les paramètres de la loi cinétique choisie dans la fenêtre précédente.
Les références sont également affichées dans un aspect plus synthétique.
Si vous souhaitez retourner à la fenêtre précédente, vous pouvez en cliquant sur le bouton « Return ».
Sinon, pour sauvegarder votre réseau de régulation biochimique, cliquez sur le bouton « Save », et une invite classique de sauvegarde de fichier vous permettra de nommer votre fichier et de le sauvegarder dans le dossier voulu.

Le fichier de suivi

Un fichier de suivi enregistre tous vos choix effectués lors des différentes étapes. Ce fichier de suivi se trouve dans le répertoire du logiciel sous le nom de « network.log ». Vous pouvez donc y retrouver tous les choix que vous avez effectués depuis votre première utilisation du logiciel. Pensez parfois à supprimer ce fichier lorsque sa taille devient trop imposante.

Annexe 5 : Spécification de la base de données PathwayDB par J.Berro

Spécifications de la base de données de réseaux de régulation

Julien Berro

2 juin 2004

Table des matières

1 Introduction **2**
1.1 Objectifs . . . 2
1.2 Qu'y a-t-il dans cette base ? . . . 2
1.3 Utilisation de la base . . . 2

2 Schéma entité-associations **3**
2.1 Sous-schéma EA concernant les interactions . . . 5
2.1.1 Entités . . . 6
2.1.2 Associations . . . 7
2.1.3 Contraintes d'intégrité . . . 8
2.2 Sous-schéma EA concernant les lieus où se passent les réactions et les conditions initiales . . . 9
2.2.1 Entités . . . 10
2.2.2 Associations . . . 10
2.3 Sous-schéma EA concernant les propriétés physico-chimiques . . . 12
2.3.1 Entités . . . 13
2.3.2 Associations . . . 13
2.3.3 Contraintes d'intégrité . . . 13

3 Code PostgreSQL **14**
3.1 Création de la base . . . 14
3.2 Contraintes d'intégrité et Triggers . . . 19

1 Introduction

1.1 Objectifs

Cette base de données a pour objectif de stocker des informations sur les interactions entre les espèces biochimiques, recoltée dans les publications scientifiques. L'idée principale est de pouvoir, à partir de cette base de donnée, reconstituer des réseaux d'interactions afin de pouvoir les simuler.

L'autre idée est de pouvoir, à partir d'un réseau reconstitué grâce à cette base, décrire facilement le réseau d'intéraction avec le langage SBML

1.2 Qu'y a-t-il dans cette base ?

La base stocke de façon cohérente les données à propos :
- des espèces biochimiques,
- des interactions entre ces molécules,
- des lois cinétiques qui les régit,
- des différentes valeurs possible des constantes de réaction,
- des lieux où se situent ces intéractions,
- des propriétés physico-chimiques des molécules (diffusion,...)
- des références bibliographiques dont sont tirées les données sus-citées.

1.3 Utilisation de la base

La base est sur la machine `applepie.imag.fr` qui estau sein du réseau de TIMC. Pour le moment, on peut s'y connecter à distance à partir de n'importe quelle machine de TIMC.

Pour vous connecter avec un client postgreSQL, il suffit de faire, à partir d'une machien de TIMC :

```
psql -h applepie.imag.fr -d PathwayDB <login>
```

2 Schéma entité-associations

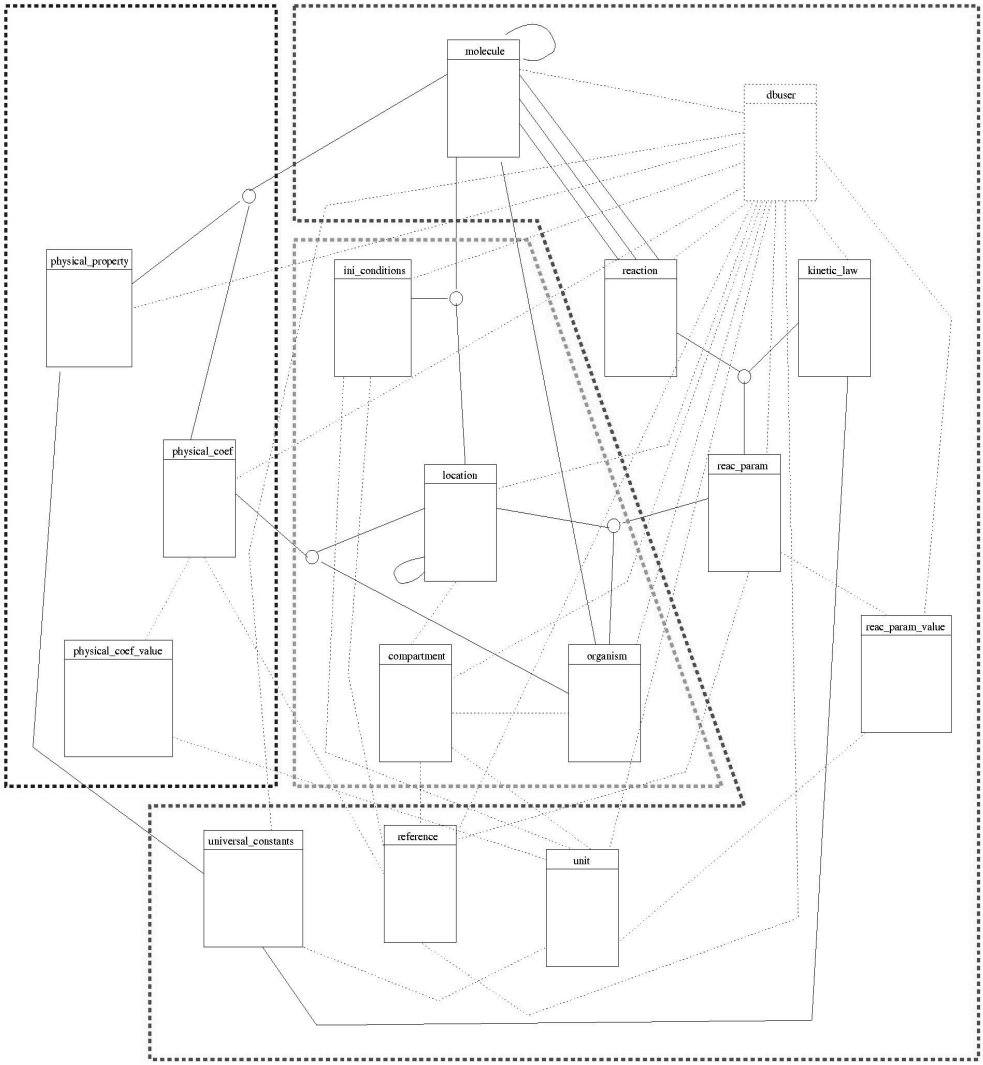


FIG. 1 – Schéma entité-association de la base de données

Tel que le schéma entité/asociation est construit, lors du passage à un schéma relationnel, les relations correspondant à chacunes des entités (sauf pour l'entité `dbuser`) auront comme clé étrangère la clé de l'entité `dbuser`, correspondant à l'utilisateur ayant rentré les données de cette entité, ainsi qu'un champ `date` correspondant à la date à laquelle les données ont été rentrées.

Remarques : dDans la description des champs des différentes entités, lorsqu'il est spécifié :

- *unique*, cela signifie qu'on ne peut avoir 2 entrées différentes ayant la même valeur pour ce champ (par exemple, si le champ `name` de l'entité `molecule` est *unique* cela signifie qu'on ne pourra pas avoir deux entrées dans la table molécule ayant le même nom).
- *optionnel*, cela signifie que ce champ peut être vide (par exemple, si le champ `charge` de l'entité `molecule` est *optionnel* cela signifie qu'on peut laisser ce champ vide)

2.1 Sous-schéma EA concernant les interactions

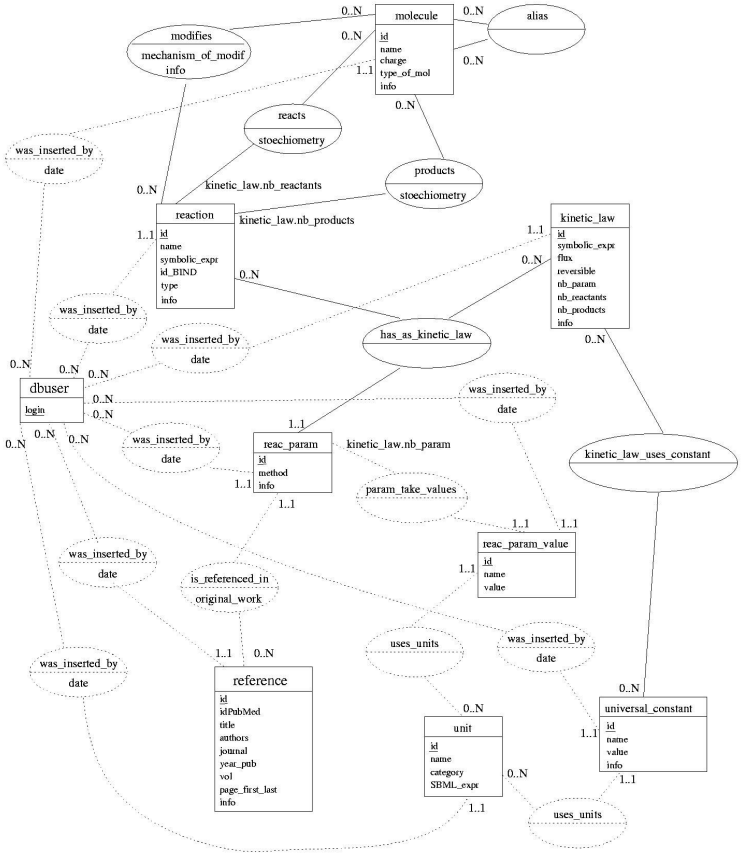


FIG. 2 – Sous-schéma concernant les intéractions (partie droite du schéma EA général)

Ce schéma décrit les entités relatifs aux intéractions.

2.1.1 Entités

Entité `molecule`
Dans l'entité `molecule` on range toutes les espèces chimiques qui peuvent réagir.
- `id (int)` est l'identifiant de la molécule.C'est la clé primaire de l'entité.
- `name (varchar)` est le nom usuel de la molecule ; unique.
- `charge (int)` est la charge que porte cette molecule (ex : $charge = +2$ pour l'ion Ca^{2+}) ; optionnel.
- `type_of_mol (set of varchar)` est le type de la molecule (ex : 'gene', 'RNA...) ; optionnel.
- `info (varchar)` contient toute autre information sur cette molecule (ex : 'protéine participant à la régulation du Calcium...') ; optionnel.

Entité `reaction` :
Cette entité contient la description qualitative des interaction, indépendament de la cinétique.
- `id (int)` est l'identifiant de la réaction.C'est la clé primaire de l'entité.
- `name (varchar)` est le nom usuel de la réaction ; unique.
- `id_BIND (varchar)` est l'identifiant de cette réaction dans la base de donnée BIND (`http ://www.blueprint.org/bind/`) ; optionnel ; unique.
- `type (varchar)` est le type de la réaction (ex : 'phosphorylation', 'hrydolysis', ...) ; optionnel.
- `info (varchar)` contient toute autre information sur cette réaction ; optionnel.

Entité `kinetic_law` :
Cette entité contient les lois cinétiques que suivent les réactions. Une loi cinétique n'est associée qu'à une réaction mais une réaction peut être associée à plusieurs lois cinétiques différentes.
- `id (int)` est l'identifiant de la loi cinétique. C'est la clé primaire de l'entité.
- `symbolic_expr (varchar)` est la loi symbolique de la réaction (ex : A+B=C).
- `flux (varchar)` expression du flux de la réaction (ex : 'K_1*(A)*(B)-K_2*(C)').
- `reversible (bool)` indique si la réaction est réversible ; par défaut : 'true'.
- `nb_param (int)` est le nombre de constantes dans l'expression de la loi cinétique (pour l'exemple de `flux` ci-dessus : `nb_param = 2`)
- `nb_reactants (int)` est le nombre de réactifs dans la loi cinétique (même exemple : `nb_reactants = 2`).
- `nb_products (int)` est le nombre de produits dans la loi cinétique (même exemple : `nb_products = 1`).
- `info (text)` contient toute autre information à propos de cette loi cinétique ; optionnel .

Remarques importantes :
- Dans `symbolic_expr` les espèces sont des lettres majuscules qui apparaissent dans l'expression dans l'ordre alphabétique (on ne peut pas avoir 'A+C=B', ni 'A+C=D', ni 'B+C=D'). Pour plus de clarté, on pose comme convention que, si, dans `symbolic_expr`, il y a '=', la réaction est supposée réversible ; s'il y a '>', elle est supposée irréversible.
- On considère toujours que les constantes de la loi cinétique présentes dans `flux` sont appelées '`K_i`' et numérotées dans l'ordre à partir de 1 jusqu'à `nb_param`, quelque soit l'unité de la constante. Si une loi cinétique nécessite une seule constante, elle sera obligatoirement nommée `K_1` ; si elle nécessite 5 constantes elles seront notées `K_1, K_2, K_3, K_4, K_5`.
- Dans l'expression du flux, les concentrations des espèces sont notées sous la forme '`( Lettre )`' (ex : K_1*(A)*(B)).

Entité `reac_param` :
Cette entité est une entité "chapeau" pour les paramètres des lois cinétiques. Elle contient des informations générales sur la manière dont ont été obtenues les valeurs des paramètres.

- `id (int)` est l'identifiant de l'entité.C'est sa clé primaire.
- `method (varchar)` est la méthode d'obtention des valeurs des paramètres (ex : 'fluorescence','mathematical model',...) ; optionnel.
- `info (text)` toute autre information concernant cette entité ; optionnel.

Entité `reac_param_value` :
Cette entité contient les valeurs des paramètres des lois cinétiques.
- `id (int)` est l'identifiant de l'entité. C'est sa clé primaire.
- `name (varchar)` est le nom du paramètre (ex : 'K_1','K'_2',... cf remarques de l'entité `kinetic_law`) .
- `value (float)` contient la valeur du paramètre,sans unité (ex : '1.2e6').

Entité `reference` :
L'entité référence contient les informations relatives aux références bibliographiques dont sont extraites toutes les données de la base.
- `id (int)` est l'identifiant de la référence. C'est la clé primaire de l'entité.
- `idPubMed (varchar)` est l'identifiant PubMed de la référence (`http ://www.ncbi.nlm.nih.gov/PubMed/`) ; optionnel ; unique.
- `title (text)` est le titre de la référence.
- `authors (text)` est la liste des auteurs.
- `journal (text)` est le nom du journal ; optionnel.
- `year_pub (int)` est l'année de publication de la référence ; optionnel.
- `vol (int)` est le numéro du volume ; optionnel.
- `page_first_last (varchar)` donne la première et la dernière page de l'article ; optionnel.
- `info (text)` toute autre information concernant la publication ; optionnel.

Entité `unit`
Cette entité référence les unités de mesure utilisées par les données chimiques et physiques
- `id (int)` est l'identifiant de l'unité. C'est la clé primaire de l'entité.
- `name (varchar)` est le nom usuel de l'unité (ex :'ţm/s') ; unique.
- `category (varchar)` est un identifiant de la catégorie à laquelle appartient l'unité (ex :'volume','temps','vitesse',...) ; optionnel.
- `SBML_expr` est l'expression de l'unité à partir de unités de base, dans le format SBML (ex :'<unit kind="metre" scale="-6"> \n <unit kind="second" exponent="-1">').

Entité `universal_constant`
Cette entité référence les constantes universelles qui sont utilisées dans différentes expressions de la base de données.
- `id (int)` est l'identifiant de la constante universelle. C'est la clé primaire de l'entité
- `name (varchar)` est le nom de la constante (ex : pi,...) ; unique.
- `value (float)` est la valeur de la constante (ex : 3.14159)
- `info (text)` : toute autre info sur la constante universelle ; optionnel

2.1.2 Associations

Association `reacts`
Cette association indique quelles molélcules interviennent comme réactifs dans une réaction donnée. Elle a comme attributs :
- `stoechiometry (int)` qui indique le nombre d'entité de cette molécule qui interviennent comme réactifs dans la réaction ; pardéfaut : 1.

Association `products`
Cette association indique quelles molélcules interviennent comme produits dans une réaction donnée. Elle a comme attributs :
- `stoechiometry (int)` qui indique le nombre d'entité de cette molécule qui interviennent comme produits dans la réaction ; pardéfaut : 1.

Association `modifies`
Cette association indique quelles molélcules interviennent comme "modifieurs" dans une réaction donnée (catalyseurs,...). Elle a comme attributs :
- `mechanism_of_modif (varchar)` qui indique le macanisme d'action de la molécule sur la réaction (ex :'catalyser', 'enzyme',...) ; optionnel.
- `info (text)` qui contient toute autre information ; optionnel.

Association `has_as_kinetic_law`
Cette association permet de relier les réactions, les lois cinétiques et les paramètres de cette loi. Lors d'une insertion, on vérifie que la loi cinétique et la réaction ont bien le même nombre de produits et de réactifs.

Association `kinetic_law_uses_constants`
Cette association permet de relier des constantes universelles avec une loi cinétique.

Association `alias`
Cette association permet de mettre en correspondance plusieurs molécules qui auraient été introduites dans la bases sous des noms différents mais qui représentent en fait une même molécule. Lors d'une insertion, on vérifie qu'un alias entre ces 2 molécules n'existe pas déjà.

Autres associations
Compte tenu de la cardinalité, les autres associations (`param_takes_values` , `uses_units` , `is_referenced_in`) n'apparaîssent pas en tant que telles dans l'expression relationnelle de la base de données. Les clés primaires des entités du côté 1..N des associations se retrouveront comme clés étrangères dans les relations issues des entités du côté 1..1 des associations. De même, les attributs des associations seront introduits dans ces dernières relations.
L'attribut `original_work` est un booléen qui vaut `true` si les valeurs des paramètres sont publiées pour la première fois dans la référence ou si elles sont uniquement citées, en référençant une publication originale.

2.1.3 Contraintes d'intégrité

- L'association `has_as_kinetic_law` est une association ternaire avec une cardinalité 1..1 . Ceci signifie que le triplet constitué par les identifiants de chacunes des entités est unique. Il ne donc pas y avoir 2 lignes égales dans la relation issue de cette association.
- Par conséquent on peut mettre la contrainte suivante : la cardinalité de l'association `param_takes_values` du côté de l'entité `reac_param` doit être égale à `kinetic_law.nb_param`. Ainsi on est sûr de donner le bon nombre de valeurs aux paramètres de la loi cinétique.
- Dans le même esprit, on veut que la loi cinétique que l'on emploie soit compatible avec la réaction à laquelle elle est associée. Elles doivent toutes les deux avoir le même nombre de réactifs et le même nombre de produits Ceci est vérifié par les cardinalités, du côté de l'entité `reaction`, des associations `products` et `reacts`.

2.2 Sous-schéma EA concernant les lieus où se passent les réactions et les conditions initiales

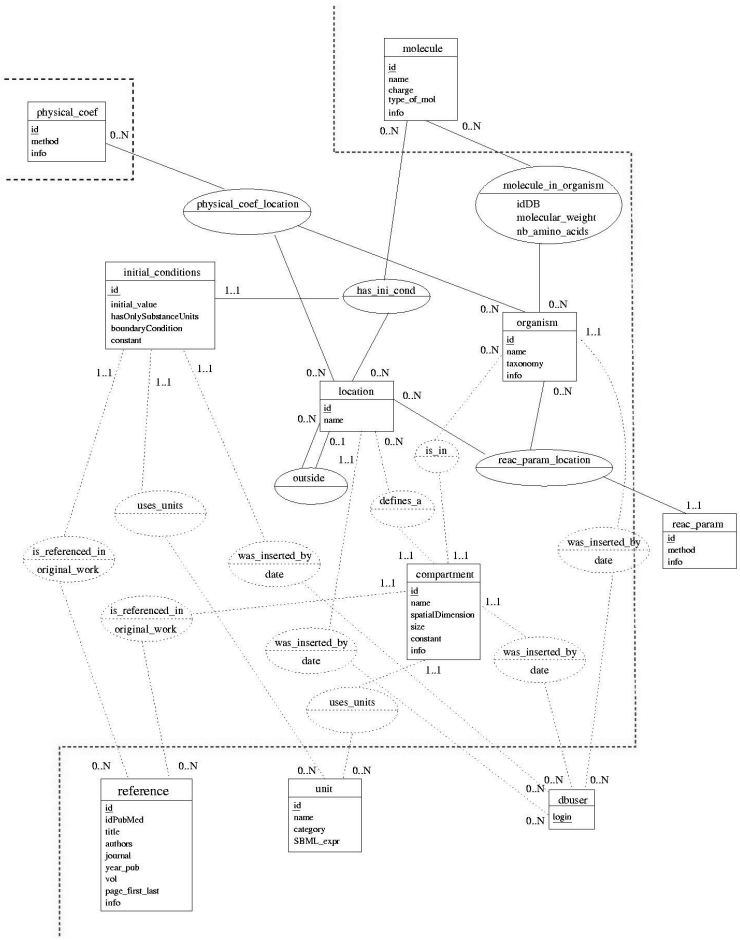


FIG. 3 – Sous-schéma EA concernant les lieus où se passent les réactions et les conditions initiales (partie centrale du schéma EA général)

2.2.1 Entités

Entité `location`
Cette entité représente le lieux biologiques où se situent les réactions, comme par exemple l'épine dendritique d'un neurone, l'ovocyte, le noyau...
- `id (int)` est l'identifiant du lieu. C'est la clé primaire de l'entité.
- `name (varchar)` est le nom du lieu (par ex : 'epine dendritique') ; unique.

Entité `compartment`
Cette entité recense des compartiment au sens où ils sont définis dans le format SBML. Ce sont des modèles géomètriques des lieux où se font les réactions.
- `id (int)` est l'identifiant du compartiment. C'est la clé primaire de l'entité.
- `name (varchar)` est le nom du compartiment ; unique.
- `spatialDimension (int)` est le nombre de dimensions de l'espace ; il est compris entre 0 et 3 ; valeur par défaut : 3 .
- `size (float)` est le volume du compartiment, sans dimension (ex :'12.4').
- `constant (bool)` indique si le compartiment a un volume constant ; valeur par défaut : 'true'.
- `info (text)` : toute autre information sur le compartiment ; optionnel.

Entité `organism`
Cette entité décrit les différents organismes dans lesquels s'effectuent les réactions
- `id (int)` est l'identifiant de l'organisme. C'est la clé primaire de l'entité.
- `name (varchar)` est le nom de l'espèce (ex : 'E. Coli','Saccharomyces Cerevisiae',...) ; unique.
- `taxonomy (varchar)` est le nom du taxon dont cette espèce fait partie (ex : Mammalia, ...).
- `info (text)` : toute autre information sur l'organisme ; optionnel.

Entité `initial_conditions`
Cette entité stocke les conditions initiales des molécules.
- `id (int)` est l'identifiant des conditions initiales. C'est la clé primaire de l'entité.
- `initial_value (float)` est la valeur initiale de la concentration , sans unité (ex : 12,3).
- `hasOnlySubstanceUnits (bool)` ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ; par défaut 'false'
- `boundaryCondition (bool)` indique si c'est une condition aux limites ; par défaut 'false'
- `constant (bool)` indique si la concentration (ou la quantité de matière) reste constante au cours des simulations ; par défaut 'false'.

2.2.2 Associations

Association `molecule_in_organism`
Cette association sert à relier une molécule avec un organisme dans lequel elle a été référencée. Ses attributs sont
- `idDB (varchar)` : identifiant d'une base de donnée sur Internet de type SwissProt ; optionnel.
- `molecular_weight (float)` : poids moleculaire en kDa ; optionnel.
- `nb_amino_acids (int)` : nombre d'acides aminés ; optionnel.

Association `has_ini_cond`
Cette association une molécule à des conditions initiales dans un lieu donné.

Association `reac_param_location`
Cette association indique dans quel organisme et dans quel lieu de cet organisme ont été évalués les jeux de paramètres des lois cinétiques.

Association `physical_coef_location`
Cette association indique dans quel organisme et dans quel lieu de cet organisme ont été évalué les coefficients des lois physico-chimiques.

Association `outside`
Cette association relie deux lieux et traduit l'inclusion d'un lieu dans un autre. Un lieu peut contenir plusieurs autres lieux mais ne peut-être contenu que dans un seul autre lieu. Plus précisément, lorsqu'un lieu est inclus dans plusieurs autres lieux, on considère qu'il est inclus uniquement dans le plus petit des lieux dans lequel il est inclus.

Remarque :
- Deux lieux ne peuvent avoir une intersection non vides sans que l'un soit inclus dans l'autre.
- Dans le schéma de la figure 4, on doit décrire les inclusions des différents lieux de la façon suivante :
  `outside( A , B )`
  `outside( B , C )`
  `outside( D , C )`
  Par exemple on n'a pas le droit d'écrire : `outside(A , C )`.

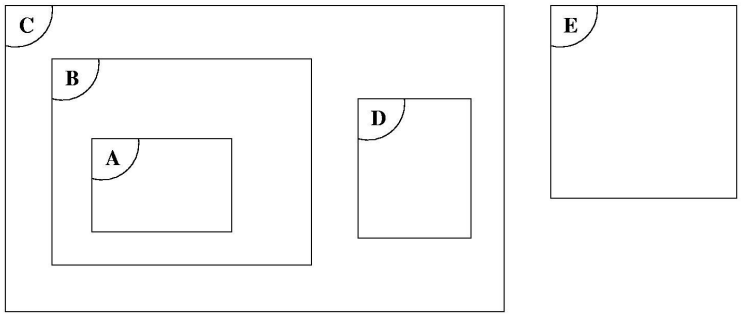


FIG. 4 – Exemple d'utilisation de la relation outside

Autres associations
Compte tenu de la cardinalité, les autres associations (`defines_a`, `uses_units`, `is_referenced_in`) n'apparaîssent pas en tant que telles dans l'expression relationnelle de la base de données. Les clés primaires des entités du côté 1..N des associations se retrouveront comme clés étrangères dans les relations issues des entités du côté 1..1 des associations. De même, les attributs des associations seront introduits dans ces dernières relations.

2.3 Sous-schéma EA concernant les propriétés physico-chimiques

Attention ! Cette partie de la bae de donnée est encore en développement... Les informations sont données à titre indicatif.

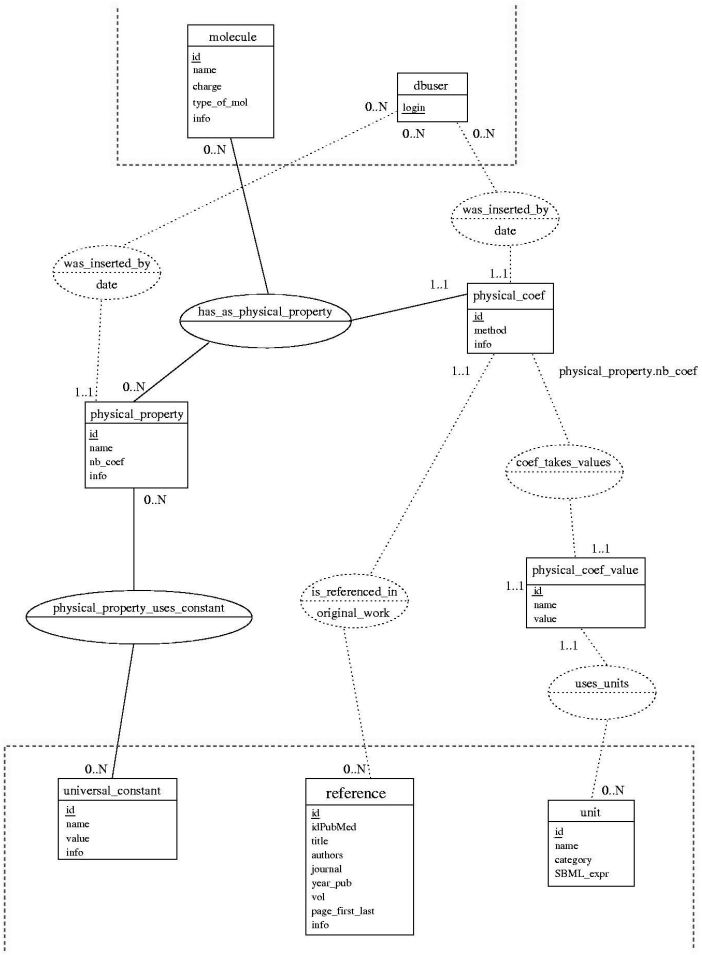


FIG. 5 – Sous-schéma EA concernant les propriétés physico-chimiques (partie de gauche du schéma EA général)

2.3.1 Entités

Entité `physical_property`
Cette entité contient les lois physico-chimiques de type diffusion, convection, ... que peuvent suivre les molécules.

- `id (int)` est l'identifiant de la propriété physico-chimique. C'est la clé primaire de l'entité.
- `name(varchar)` est le nom de la propriété ; unique.
- `law (text)` est la loi cinétique suivie.
- `nb_coef (int)` est le nombre de coefficients décrivant cette propriété.
- `info (text)` : toute autre information sur la propriété physico-chimique ; optionnel.

Entité `physical_coef`
Cette entité est une entité chapeau qui regroupe l'ensemble des paramètres des propriétés physico-chimiques

- `id (int)` est l'identifiant des paramètres physico-chimiques. C'est la clé primaire de l'entité.
- `method (varchar)` est la méthode employée pour obtenir les paramètres .
- `info (text)` : toute autre information à propos des paramètres ; optionnel

Entité `physical_coef_values`
Cette entité contient les valeurs des paramètres des propriétés physico-chimiques

- `id (int)` est l'identifiant du paramètre. C'est la clé primaire de l'entité.
- `name (varchar)` est le nom du paramètre .
- `value (float)` est la valeur du paramètre, sans unité (ex : 12.3).

2.3.2 Associations

Association `has_as_physical_property`
Cette association relie une propriété physico-chimique avec les lois physiques à une molécule et aux paramètres qui la régit.

Association `physical_property_uses_constants`
Cette association relie une propriété physico-chimique avec les consatantes universelles employées dans cette loi.

Autres associations
Compte tenu de la cardinalité, les autres associations (`defines_a , uses_units , is_referenced_in`) n'apparaîssent pas en tant que telles dans l'expression relationnelle de la base de données. Les clés primaires des entités du côté 1..N des associations se retrouveront comme clés étrangères dans les relations issues des entités du côté 1..1 des associations.

2.3.3 Contraintes d'intégrité

La cardinalité de l'association `coef_take_values` du côté de l'entité `physical_coef` doit être égal à la valeur du champ `nb_coef` de l'entité `physical_property`. Ceci permet d'avoir de donner le bon nombre de valeurs aux paramètres des loi physico-chimiques.

3 Code PostgreSQL

3.1 Création de la base

```
-- select * from pg_tables where schemaname='public';

DROP TABLE dbuser,physical_coef_value,physical_coef,physical_property,
   molecule,universal_constant,unit,reference,initial_conditions,
   organism,compartment,location,reac_param_value,reac_param,
   kinetic_law,reaction,physical_property_uses_constant,
   has_as_physical_property,has_ini_cond,reacts,products,modifies,
   alias,molecule_in_organism,outside,physical_coef_location,
   reac_param_location,kinetic_law_uses_constant,has_as_kinetic_law
   CASCADE;

--------------------
---- ENTITES -------
--------------------
-- Utilisateurs
CREATE TABLE dbuser(
  login varchar(100) UNIQUE,
  PRIMARY KEY (login)
);

-- References bibliographiques
CREATE TABLE reference (
  id serial ,
  id_Pubmed varchar(100) UNIQUE,
  title varchar(100) NOT NULL,
  authors varchar(100) NOT NULL,
  journal varchar(100),
  year_pub int8,
  vol int8,
  page_first_last varchar(100),
  info TEXT,
  login_user varchar(100) NOT NULL REFERENCES dbuser(login),
  date timestamp NOT NULL,
-- date de type 1999-01-08 04:05:06
  PRIMARY KEY  (id)
);


-- Constantes / Unites

CREATE TABLE unit(
  id serial  ,
  name varchar(100) NOT NULL UNIQUE,
  category varchar(100),
  SBML_expr TEXT NOT NULL,
  login_user varchar(100) NOT NULL REFERENCES dbuser(login),
  date timestamp NOT NULL,
  PRIMARY KEY  (id)
);

CREATE TABLE universal_constant (
  id serial  ,
  name varchar(100) NOT NULL UNIQUE,
  value float NOT NULL,
  info TEXT,
  id_unit INT8  NOT NULL REFERENCES unit(id),
  login_user varchar(100) NOT NULL REFERENCES dbuser(login),
  date timestamp NOT NULL,
  PRIMARY KEY  (id)
);


-- Molecules
CREATE TABLE molecule(
  id serial NOT NULL PRIMARY KEY,
  name varchar(100) NOT NULL UNIQUE,
  charge int8,
  type_of_mol varchar(100) CHECK( type_of_mol in ('protein/generic',
        'protein/receptor','protein/ion channel','protein/truncated',
        'gene','RNA','antisense RNA','phenotype','ion','simple molecule',
        'unknown','heterodimer','other')),
  info text,
  login_user varchar(100) NOT NULL REFERENCES dbuser(login),
  date timestamp NOT NULL
```

```
);


-- Physique
CREATE TABLE physical_property (
  id serial  ,
  name varchar(100) NOT NULL UNIQUE,
  nb_coef varchar(100) NOT NULL,
  info text,
  login_user varchar(100) NOT NULL REFERENCES dbuser(login),
  date timestamp NOT NULL,
  PRIMARY KEY  (id)
);

CREATE TABLE physical_coef (
  id serial ,
  method varchar(100),
  info TEXT,
  id_ref INT8  NOT NULL REFERENCES reference(id),
  original_work BOOL NOT NULL,
  login_user varchar(100) NOT NULL REFERENCES dbuser(login),
  date timestamp NOT NULL,
  PRIMARY KEY  (id)
);

CREATE TABLE physical_coef_value(
  id serial  ,
  name varchar(100) NOT NULL,
  value float NOT NULL,
  id_physical_coef INT8  NOT NULL REFERENCES physical_coef(id),
  id_unit INT8  NOT NULL REFERENCES unit(id),
  PRIMARY KEY (id)
);


-- Chimie
CREATE TABLE reaction (
  id serial  ,
  name varchar(200) NOT NULL UNIQUE,
  symbolic_expr varchar(100) NOT NULL UNIQUE,
  id_BIND varchar(100) UNIQUE,
  type varchar(100) CHECK( type in ('transcriptional activation',
        'transcriptional inhibition','translational activation',
        'translational inhibition','catalysis','inhibition','state transition',
        'known transition omitted','unknown transition','transport',
        'heterodimer formation','dissociation','truncation',
        'homodimer formation','degradation')),
  info text,
  login_user varchar(100) NOT NULL REFERENCES dbuser(login),
  date timestamp NOT NULL,
  PRIMARY KEY  (id)
);

CREATE TABLE kinetic_law (
  id serial  ,
--  name varchar(100) NOT NULL UNIQUE,
  symbolic_expr varchar(100) NOT NULL,
  flux text NOT NULL,
  reversible BOOL default 'TRUE',
  nb_param INT8  NOT NULL,
  nb_reactants int8 not null,
  nb_products int8 not null,
  info TEXT,
  login_user varchar(100) NOT NULL REFERENCES dbuser(login),
  date timestamp NOT NULL,
  PRIMARY KEY  (id)
);

CREATE TABLE reac_param (
  id serial  ,
  method varchar(100),
  info TEXT,
  id_ref INT8  NOT NULL REFERENCES reference(id),
  original_work BOOL,
  login_user varchar(100) NOT NULL REFERENCES dbuser(login),
  date timestamp NOT NULL,
  PRIMARY KEY  (id)
);

CREATE TABLE reac_param_value (
```

```
  id serial,
  name varchar(100) NOT NULL,
--il faudrait verifier que le nom existe bien dans la loi cinetique
  value float NOT NULL,
  id_reac_param INT8  NOT NULL REFERENCES reac_param(id),
  id_unit INT8  NOT NULL REFERENCES unit(id),
  --login_user varchar(100) NOT NULL REFERENCES dbuser(login),
  --date timestamp NOT NULL,
  PRIMARY KEY (id)
);
ALTER TABLE reac_param_value
       ADD CONSTRAINT unique_constraint_reac_param_value UNIQUE (id_reac_param,name);
-- we can't have 2 parameters with the same name for one reac_param


-- Compartiments / Organisme
CREATE TABLE location (
  id serial  ,
  name varchar(100) NOT NULL UNIQUE,
  login_user varchar(100) NOT NULL REFERENCES dbuser(login),
  date timestamp NOT NULL,
  PRIMARY KEY  (id)
);

CREATE TABLE organism (
  id serial  ,
  name varchar(100) NOT NULL UNIQUE,
  taxonomy varchar(100) NOT NULL,
  info TEXT,
  login_user varchar(100) NOT NULL REFERENCES dbuser(login),
  date timestamp NOT NULL,
  PRIMARY KEY  (id)
);

CREATE TABLE compartment (
  id serial  ,
  name varchar(100) NOT NULL UNIQUE,
  spatialDimension int2 NOT NULL check (spatialDimension in ('0','1','2','3')) default '3',
  size FLOAT NOT NULL,
  constant BOOL NOT NULL default 'TRUE',
  info TEXT,
  id_location INT8  NOT NULL REFERENCES location(id),
  id_organism INT8  NOT NULL REFERENCES organism(id),
  id_unit INT8  NOT NULL REFERENCES unit(id),
  id_ref INT8  NOT NULL REFERENCES reference(id),
  original_work BOOL,
  login_user varchar(100) NOT NULL REFERENCES dbuser(login),
  date timestamp NOT NULL,
  PRIMARY KEY  (id)
);

CREATE TABLE initial_conditions (
  id serial NOT NULL PRIMARY KEY,
  initial_value float NOT NULL,
  hasOnlySubstanceUnits BOOL NOT NULL default 'FALSE',
  boundaryCondition BOOL NOT NULL default 'FALSE',
  constant BOOL NOT NULL default 'FALSE',
  id_unit INT8 NOT NULL REFERENCES unit(id),
  id_ref INT8 NOT NULL REFERENCES reference(id),
  original_work BOOL,
  login_user varchar(100) NOT NULL REFERENCES dbuser(login),
  date timestamp NOT NULL
);




---------------------
--- ASSOCIATIONS ----
---------------------

-- Molecule/Reaction

CREATE TABLE alias(
  id_mol1 INT8 NOT NULL REFERENCES molecule(id) ON DELETE CASCADE,
  id_mol2 INT8 NOT NULL REFERENCES molecule(id) ON DELETE CASCADE
);-- trigger_alias checks that aliases are referenced only once in table alias
```

```
CREATE TABLE modifies (
  id_mol INT8  NOT NULL REFERENCES molecule(id) ON DELETE CASCADE,
  id_reac INT8  NOT NULL REFERENCES reaction(id) ON DELETE CASCADE,
  mechanism_of_modif varchar(100) ,
  info TEXT,
  UNIQUE(id_mol,id_reac)
);

CREATE TABLE products (
  id_mol INT8  NOT NULL REFERENCES molecule(id) ON DELETE CASCADE,
  id_reac INT8  NOT NULL REFERENCES reaction(id) ON DELETE CASCADE,
  stoechiometry INT8  NOT NULL default '1',
  UNIQUE(id_mol,id_reac)
);

CREATE TABLE reacts (
  id_mol INT8  NOT NULL REFERENCES molecule(id) ON DELETE CASCADE,
  id_reac INT8  NOT NULL REFERENCES reaction(id) ON DELETE CASCADE,
  stoechiometry INT8  NOT NULL default '1',
  UNIQUE(id_mol,id_reac)
);

CREATE TABLE has_ini_cond (
  id_mol INT8  NOT NULL REFERENCES molecule(id) ON DELETE CASCADE,
  id_ini_cond INT8  NOT NULL REFERENCES initial_conditions(id) ON DELETE CASCADE,
  id_location INT8  NOT NULL REFERENCES location(id) ON DELETE CASCADE,
  UNIQUE(id_ini_cond)
);



-- Mecanique
CREATE TABLE has_as_physical_property (
  id_physical_prop INT8  NOT NULL REFERENCES physical_property(id) ON DELETE CASCADE,
  id_physical_coef INT8  NOT NULL REFERENCES physical_coef(id) ON DELETE CASCADE,
  id_mol INT8  NOT NULL REFERENCES molecule(id) ON DELETE CASCADE,
  UNIQUE(id_physical_coef)
);

CREATE TABLE physical_property_uses_constant (
  id_physical_prop INT8  NOT NULL REFERENCES physical_property(id) ON DELETE CASCADE,
  id_constant INT8  NOT NULL REFERENCES universal_constant(id) ON DELETE CASCADE,
  UNIQUE(id_physical_prop,id_constant)
);


-- Chimie
CREATE TABLE has_as_kinetic_law (
  id_reac INT8  NOT NULL REFERENCES reaction(id) ON DELETE CASCADE,
  id_kinetic_law INT8  NOT NULL REFERENCES kinetic_law(id) ON DELETE CASCADE,
  id_reac_param INT8 UNIQUE NOT NULL REFERENCES reac_param(id) ON DELETE CASCADE
  -- we check that the kinetic_law and the reaction has the same number of reactants
  -- and products with the trigger 'check_reaction_kinetic_law'
);


CREATE TABLE kinetic_law_uses_constant (
  id_kinetic_law INT8  NOT NULL REFERENCES kinetic_law(id) ON DELETE CASCADE,
  id_constant INT8  NOT NULL REFERENCES universal_constant(id) ON DELETE CASCADE,
  UNIQUE(id_kinetic_law,id_constant)
);

-- Lieu / Compartiment / Organisme
CREATE TABLE reac_param_location (
  id_organism INT8  NOT NULL REFERENCES organism(id) ON DELETE CASCADE,
  id_location INT8  NOT NULL REFERENCES location(id) ON DELETE CASCADE,
  id_reac_param INT8  NOT NULL REFERENCES reac_param(id) ON DELETE CASCADE,
  UNIQUE(id_reac_param)
);

CREATE TABLE physical_coef_location (
  id_physical_coef INT8  NOT NULL REFERENCES physical_coef(id) ON DELETE CASCADE,
  id_location INT8  NOT NULL REFERENCES location(id) ON DELETE CASCADE,
  id_organism INT8  NOT NULL REFERENCES organism(id) ON DELETE CASCADE,
  UNIQUE(id_physical_coef)
);


CREATE TABLE outside (
```

```
--location2 is just outside location1
-- i.e. location1 is included in location2
  id_location1 INT8  NOT NULL REFERENCES location(id) ON DELETE CASCADE,
  id_location2 INT8  NOT NULL REFERENCES location(id) ON DELETE CASCADE,
  UNIQUE(id_location1)
);

CREATE TABLE molecule_in_organism (
  id_molecule INT8  NOT NULL REFERENCES molecule(id) ON DELETE CASCADE,
  id_organism INT8  NOT NULL REFERENCES organism(id) ON DELETE CASCADE,
  idDB varchar(100),
  molecular_weight float,
  nb_amino_acids int,
  UNIQUE(id_molecule,id_organism)
);
```

3.2 Contraintes d'intégrité et Triggers

```
--Initialisation
DROP FUNCTION plpgsql_call_handler() CASCADE;
  --DROP LANGUAGE 'plpgsql';

CREATE FUNCTION plpgsql_call_handler()
  RETURNS LANGUAGE_HANDLER
  AS '/usr/lib/pgsql/plpgsql.so' --bibliotheque de plpgsql -> sous linux plpgsql.so
  LANGUAGE 'C';

CREATE LANGUAGE 'plpgsql' HANDLER plpgsql_call_handler LANCOMPILER 'PL/pgSQL';


-- Fonctions
CREATE FUNCTION check_nb_reactants_reaction_kinetic_law(int8,int8) RETURNS bool AS'
-- We check that the kinetic_law and the reaction has the same number of reactants
  DECLARE
    id_reac ALIAS FOR $1;
    id_kinetic_law ALIAS FOR $2;
    chk bool;
    nb_reactants_reaction int8;
    nb_reactants_kinetic_law int8;
    name_reac text;
    name_kinetic_law text;
  BEGIN
    select into nb_reactants_reaction sum(stoechiometry)
        from reacts --,has_as_kinetic_law
        where reacts.id_reac=id_reac
        group by reacts.id_reac;
    select into nb_reactants_kinetic_law nb_reactants
        from kinetic_law --,has_as_kinetic_law
        where kinetic_law.id=id_kinetic_law;
    if nb_reactants_reaction=nb_reactants_kinetic_law then
        chk:=true;
    else
        chk:=false;
        select into name_kinetic_law symbolic_expr from kinetic_law where id=id_kinetic_law;
        select into name_reac symbolic_expr from reaction where id=id_reac;
        RAISE EXCEPTION '' This kinetic law % does not fit the reaction %:
                        number of reactants does not match \n in reaction: % \n in kinetic law : % \n id_reac=% '',
                        name_kinetic_law,name_reac,nb_reactants_reaction , nb_reactants_kinetic_law,id_reac;
    end if;
    return chk;
  END;
' LANGUAGE 'plpgsql';

CREATE FUNCTION check_nb_products_reaction_kinetic_law(int8,int8) RETURNS bool AS'
-- We check that the kinetic_law and the reaction has the same number of products
  DECLARE
    chk bool;
    nb_products_reaction int8;
    nb_products_kinetic_law int8;
    id_reac ALIAS FOR $1;
    id_kinetic_law ALIAS FOR $2;
    name_reac text;
    name_kinetic_law text;
  BEGIN
    select into nb_products_reaction sum(stoechiometry)
        from products --,has_as_kinetic_law
--      where (products.id_reac=has_as_kinetic_law.id_reac and products.id_reac=id_reac)
        where products.id_reac=id_reac
        group by products.id_reac;
    select into nb_products_kinetic_law nb_products
        from kinetic_law --,has_as_kinetic_law
        where kinetic_law.id=id_kinetic_law;
    if nb_products_reaction=nb_products_kinetic_law then
        chk:=true;
    else
        chk:=false;
        select into name_kinetic_law symbolic_expr from kinetic_law where id=id_kinetic_law;
        select into name_reac symbolic_expr from reaction where id=id_reac;
        RAISE EXCEPTION '' This kinetic law % does not fit the reaction % : number of products does not match
                        \n in reaction: % \n in kinetic law : %'',
                        name_kinetic_law,name_reac,nb_products_reaction , nb_products_kinetic_law;
    end if;
    return chk;
  END;
' LANGUAGE 'plpgsql';
```

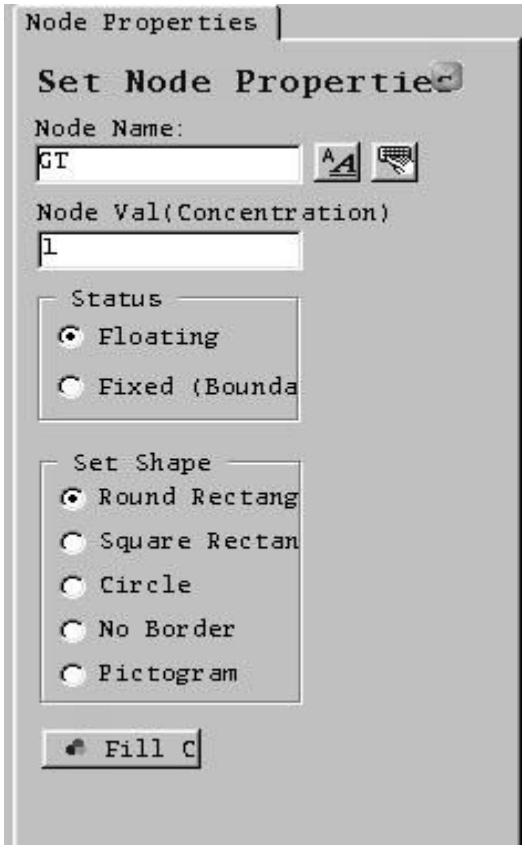
```
CREATE FUNCTION check_reaction_kinetic_law_compatibility() RETURNS TRIGGER AS'
-- We check that the kinetic_law and the reaction has the same
-- number of reactants and products
  DECLARE
    id_reac int8;
    id_kinetic_law int8;
    a bool;
  BEGIN
    select into id_kinetic_law kinetic_law.id
        from kinetic_law
        where kinetic_law.id=NEW.id_kinetic_law;
    if not found then
        raise exception ''INTERN: NEW is empty 1'';
    end if;
    select into id_reac reaction.id
        from reaction
        where reaction.id=NEW.id_reac;
    if not found then
        raise exception ''INTERN: NEW is empty'';
    end if;
    a:=check_nb_reactants_reaction_kinetic_law(id_reac,id_kinetic_law);
    a:=check_nb_products_reaction_kinetic_law(id_reac,id_kinetic_law);
    return NEW;
  END;
' LANGUAGE 'plpgsql';

-- Triggers
CREATE TRIGGER check_reaction_kinetic_law
-- We check that the kinetic_law and the reaction has the same
-- number of reactants and products
  AFTER INSERT OR UPDATE
  ON has_as_kinetic_law FOR EACH ROW -- ROW could be better but slower
  EXECUTE PROCEDURE check_reaction_kinetic_law_compatibility();


-------------------------------------------------------------------------------

-- We check that number in reac_param_values is less than the
-- number of parameter in the kinetic_law
CREATE FUNCTION check_kinetic_law_reac_param_value_compatibility() RETURNS TRIGGER AS'
-- we check that number in reac_param_values is less than the number of parameter in the kinetic_law
  DECLARE
    nb_param_in_param_reac int8;
    nb_param_in_kinetic_law int8;
  BEGIN
    select into nb_param_in_param_reac count(*)
        from reac_param_value
        where reac_param_value.id_reac_param=NEW.id_reac_param;
    select into nb_param_in_kinetic_law nb_param
        from kinetic_law
        where kinetic_law.id=(select id_kinetic_law from has_as_kinetic_law
                        where has_as_kinetic_law.id_reac_param=NEW.id_reac_param);
    if nb_param_in_param_reac >  nb_param_in_kinetic_law then
        RAISE EXCEPTION '' There are to many parameters for the kinetic law \n
        The kinetic law needs % parameter(s) whereas you are trying
        to give % parameters'',nb_param_in_kinetic_law, nb_param_in_param_reac;
    end if;
    return NEW;
  END;
' LANGUAGE 'plpgsql';

CREATE TRIGGER check_kinetic_law_reac_param_value
-- We check that number in reac_param_values is less than
-- the number of parameter in the kinetic_law
  AFTER INSERT OR UPDATE
  ON reac_param_value FOR EACH ROW -- ROW could be better but slower
  EXECUTE PROCEDURE check_kinetic_law_reac_param_value_compatibility();

-------------------------------------------------------------------------------------------

-- We check that molecule id_mol belongs to reaction id_reac
CREATE FUNCTION check_has_ini_cond_compatibility() RETURNS TRIGGER AS'
-- If molecule id_mol belongs to reaction id_reac then
-- it exists one ligne in reacts or in products whitch contains id_mol and id_reac

  DECLARE
    nb_mol_in_reacts int8;
    nb_mol_in_products int8;
    mol_name varchar(100);
    reac_expr varchar(100);
```

```
    aa int8;
  BEGIN
    select into nb_mol_in_reacts sum(stoechiometry)
        from reacts
        where (reacts.id_mol=NEW.id_mol and reacts.id_reac=NEW.id_reac);
    select into nb_mol_in_products sum(stoechiometry)
        from products
        where (products.id_mol=NEW.id_mol and products.id_reac=NEW.id_reac);
    if (nb_mol_in_reacts>0 or nb_mol_in_products>0) then
          return NEW;
    else
        select into mol_name name from molecule where molecule.id=NEW.id_mol;
        select into reac_expr symbolic_expr from reaction where reaction.id=NEW.id_reac;
        aa=nb_mol_in_reacts+nb_mol_in_products;
        RAISE EXCEPTION '' Molecule % does not belongs to reaction % . So, you can not associate these initial conditions with this mole
               mol_name,reac_expr,aa;
    end if;
  END;
' LANGUAGE 'plpgsql';

--CREATE TRIGGER trigger_has_ini_cond
-- We check that molecule id_mol belongs to reaction id_reac
--  AFTER INSERT OR UPDATE
--  ON has_ini_cond FOR EACH ROW
--  EXECUTE PROCEDURE check_has_ini_cond_compatibility();

-------------------------------------------------------------------------------------------------

-- We check that in alias, there is only one alias when 2 molecules have the same name
CREATE FUNCTION check_alias_unicity() RETURNS TRIGGER AS'
  DECLARE
    mol1 int8;
    mol2 int8;
    name_mol1 varchar(100);
    name_mol2 varchar(100);
    temp1 int8;
    temp2 int8;
  BEGIN
    select into mol1 NEW.id_mol1;
    select into mol2 NEW.id_mol2;
    select into temp1 id_mol1 from alias
        where (id_mol1=mol1 and id_mol2=mol2);
    select into temp2 id_mol1 from alias
        where (id_mol1=mol2 and id_mol2=mol1);
    if ( (temp1 IS NOT NULL) OR (temp2 IS NOT NULL) ) then
        select into name_mol1 name from molecule
          where id=mol1;
        select into name_mol2 name from molecule
          where id=mol2;
        RAISE EXCEPTION '' Molecules % and % are yet known to represent the same molecule,
                        ie (%,%) is yet referenced in table alias \n % ; %'',
               name_mol1,name_mol2,mol1,mol2,temp1,temp2;
    end if;
    if ( mol1=mol2 ) then
        select into name_mol1 name from molecule
          where id=mol1;
        RAISE EXCEPTION '' You are trying to insert an alias between 2 molecules who have the same name :%'',name_mol1;
    end if;
    return NEW;
  END;
' LANGUAGE 'plpgsql';

CREATE TRIGGER trigger_alias
-- We check that molecule id_mol belongs to reaction id_reac
  BEFORE INSERT OR UPDATE
  ON alias FOR EACH ROW
  EXECUTE PROCEDURE check_alias_unicity();

-------------------------------------------------------------------------------------------------------
```

Annexe 6 : Fenêtre de JDesigner

La fenêtre de JDesigner est composé de trois éléments : une partie contenant les propriétés de la molécule sélectionnée, une partie contenant les propriétés de la réactions sélectionnée, puis une partie montrant un schéma du réseau créé.

Propriétés des molécules.



Propriétés des réactions.

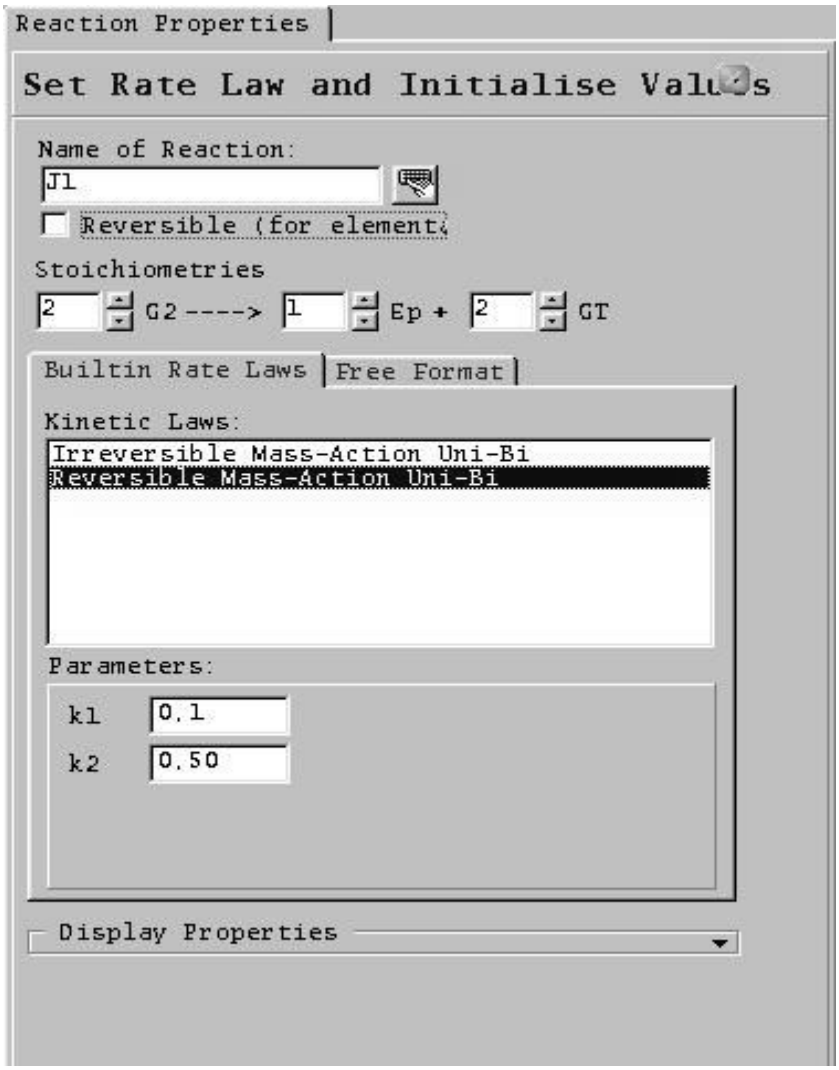


Schéma du réseau: